UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES
CENTRO DE CIENCIAS BÁSICAS

TESIS

DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE RIEGO INTELIGENTE BASADO EN INTERNET DE LAS COSAS PARA CULTIVOS PROTEGIDOS EN AGUASCALIENTES, MÉXICO

PRESENTA

Ing. William Alejandro Angulo Martínez

PARA OBTENER EL GRADO DE MAESTRÍA EN CIENCIAS CON OPCIONES A LA COMPUTACIÓN, MATEMÁTICAS APLICADAS

TUTOR

Dr. Julio César Ponce Gallegos

ASESORES

Dr. Alejandro Padilla Díaz

Dr. Jaime Muñoz Arteaga

Aguascalientes, Ags, Noviembre de 2025

CARTA DE VOTO APROBATORIO

MTRO. EN C. JORGE MARTÍN ALFÉREZ CHÁVEZ
DECANO DEL CENTRO DE CIENCIAS BÁSICAS

P R E S E N T E

Por medio del presente como **DIRECTOR** designado del estudiante **WILLIAM ALEJANDRO ANGULO MARTÍNEZ** con ID 538560 quien realizó la tesis titulada: **DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE RIEGO INTELIGENTE BASADO EN INTERNET DE LAS COSAS PARA CULTIVOS PROTEGIDOS EN AGUASCALIENTES, MÉXICO**, un trabajo propio, innovador, relevante e inédito y con fundamento en la fracción IX del Artículo 43 del Reglamento General de Posgrados, doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que él pueda continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

A T E N T A M E N T E
"Se Lumen Proferre"
Aguascalientes, Ags., a 12 de noviembre de 2025.

Dr. Julio César Ponce Gallegos
Director de tesis

c.c.p.- Interesado
c.c.p.- Coordinación del Programa de Posgrado

CARTA DE VOTO APROBATORIO

MTRO. EN C. JORGE MARTÍN ALFÉREZ CHÁVEZ
DECANO DEL CENTRO DE CIENCIAS BÁSICAS

PRESENTE

Por medio del presente como **ASESOR** designado del estudiante **WILLIAM ALEJANDRO ANGULO MARTÍNEZ** con ID 538560 quien realizó la tesis titulada: **DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE RIEGO INTELIGENTE BASADO EN INTERNET DE LAS COSAS PARA CULTIVOS PROTEGIDOS EN AGUASCALIENTES, MÉXICO**, un trabajo propio, innovador, relevante e inédito y con fundamento en la fracción IX del Artículo 43 del Reglamento General de Posgrados, doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que él pueda continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 12 de noviembre de 2025.

Dr. Alejandro Padilla Díaz
Asesor de tesis

c.c.p.- Interesado
c.c.p.- Coordinación del Programa de Posgrado

CARTA DE VOTO APROBATORIO

MTRO. EN C. JORGE MARTÍN ALFÉREZ CHÁVEZ
DECANO DEL CENTRO DE CIENCIAS BÁSICAS

PRESENTE

Por medio del presente como **ASESOR** designado del estudiante **WILLIAM ALEJANDRO ANGULO MARTÍNEZ** con ID 538560 quien realizó la tesis titulada: **DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE RIEGO INTELIGENTE BASADO EN INTERNET DE LAS COSAS PARA CULTIVOS PROTEGIDOS EN AGUASCALIENTES, MÉXICO,** un trabajo propio, innovador, relevante e inédito y con fundamento en la fracción IX del Artículo 43 del Reglamento General de Posgrados, doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que él pueda continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 12 de noviembre de 2025.

Dr. Jaime Muñoz Arteaga
Asesor de tesis

c.c.p.- Interesado
c.c.p.- Coordinación del Programa de Posgrado

Universidad Autónoma de Aguascalientes

DICTAMEN DE LIBERACIÓN ACADÉMICA PARA INICIAR LOS TRÁMITES DEL EXAMEN DE GRADO

POSGRADOS UAA

Fecha de dictaminación (dd/mm/aaaa): 21/11/2025

NOMBRE: William Alejandro Angulo Martínez **ID** 538560

PROGRAMA: Maestría en Ciencias con Opciones a la Computación, Matemáticas Aplicadas **LGAC (del posgrado):** Ingeniería de Software

MODALIDAD DEL PROYECTO DE GRADO: Tesis Tradicional (X) *Tesis por artículos científicos ( ) **Tesis por Patente ( ) Trabajo Práctico ( )

TITULO: Desarrollo e Implementación de un Sistema de Riego Inteligente Basado en IoT para Cultvios Protegidos en Aguascalientes, México

IMPACTO SOCIAL (señalar el impacto logrado): Se trabajo para ayudar al cuidado del agua en el estado, ya que es importante porque es un recurso escaso y que traerá problemas en un futuro cercano en produccion de alimento, agua potable para la poblacion y como materia prima(recurso indispensable) de varias industrias

INDICAR SEGÚN CORRESPONDA: **SI, NO, NA** (No Aplica)

	Elementos para la revisión académica del trabajo de tesis o trabajo práctico:
SI	El trabajo es congruente con las LGAC del programa de posgrado
SI	La problemática fue abordada desde un enfoque multidisciplinario
SI	Existe coherencia, continuidad y orden lógico del tema central con cada apartado
SI	Los resultados del trabajo dan respuesta a las preguntas de investigación o a la problemática que aborda
SI	Los resultados presentados en el trabajo son de gran relevancia científica, tecnologica o profesional según el área
SI	El trabajo demuestra más de una aportación original al conocimiento de su área
SI	Las aportaciones responden a los problemas prioritarios del país
SI	Generó transferencia del conocimiento o tecnológica
SI	Cumple con la ética para la investigación (reporte de la herramienta antiplagio)
	El egresado cumple con lo siguiente:
SI	Cumple con lo señalado por el Reglamento General de Posgrados
SI	Cumple con los requisitos señalados en el plan de estudios (créditos curriculares, optativos, actividades complementarias, estancia, predoctoral, etc.)
SI	Cuenta con los votos aprobatorios del comité tutorial
NA	Cuenta con la carta de satisfacción del Usuario (En caso de que corresponda)
SI	Coincide con el título y objetivo registrado
SI	Tiene congruencia con cuerpos académicos
SI	Tiene el CVU de la SECIHTI actualizado
NA	Tiene el o los artículos aceptados o publicados y cumple con los requisitos institucionales (en caso de que proceda)
	****En caso de Tesis por artículos científicos publicados*** *(completar solo si la tesis fue por artículos)*
NA	Aceptación o Publicación de los artículos en revistas indexadas de alto impacto según el nivel del programa
NA	El (la) estudiante es el primer autor(a)
NA	El (la) autor(a) de correspondencia es el Director (a) del Núcleo Académico
NA	En los artículos se ven reflejados los objetivos de la tesis, ya que son producto de este trabajo de investigación.
NA	Los artículos integran los capítulos de la tesis y se presentan en el idioma en que fueron publicados
	*****En caso de Tesis por Patente***
NA	Cuenta con la evidencia de solicitud de patente en el Departamento de Investigación (anexarla al presente formato)

Con base en estos criterios, se autoriza continuar con los trámites de titulación y programación del examen de grado: **Sí** X **No** ____

FIRMAS

Elaboró:

*NOMBRE Y FIRMA DEL(LA) CONSEJERO(A) SEGÚN LA LGAC DE ADSCRIPCION: P.A. Dr. Jaime Muñoz Arteaga

** En caso de conflicto de intereses, firmará un revisor miembro del NA de la LGAC correspondiente distinto al director o miembro del comité tutorial, asignado por el Decano.*

NOMBRE Y FIRMA DEL COORDINADOR DE POSGRADO: Dra. Mariana Alfaro Gómez

Revisó:

NOMBRE Y FIRMA DEL SECRETARIO DE INVESTIGACIÓN Y POSGRADO: Dr. Alejandro Padilla Díaz

Autorizó:

NOMBRE Y FIRMA DEL DECANO: M. en. C. Jorge Martín Alférez Chávez

Nota: procede el trámite para el Depto. de Apoyo al Posgrado

En cumplimiento con el Art. 24 fracción V del Reglamento General de Posgrado, que a la letra señala entre las funciones del Consejo Académico: Proponer criterios y mecanismos de selección, permanencia, egreso y titulación de estudiantes para asegurar la eficiencia terminal y la titulación y el Art. 28 fracción IX, atender, asesorar y dar el seguimiento del estudiantado desde su ingreso hasta su titulación.

Autorización para la publicación de Tesis electrónicas

Fecha 11/24/2025

1) Datos personales

Nombre	William Alejandro Angulo Martínez		
Dirección	Prolongación Libertad 1727 Int. 6	C.P.	20020
Ciudad	Aguascalientes	Estado	Aguascalientes
Teléfono	5559076323	E-Mail	williamangulomartinez@gmail.com
Grado Académico actual	Licenciatura		
CURP(18 car.)	AUMW940617HNENRL08		
RFC(13 car.)	AUMW940617IA1		
ORCID(16 car.)	0009-0000-7370-0647		

Si Usted cuenta con un registro ORCID, le pedimos atentamente lo proporcione.

2)Datos escolares

Centro	Centro de Ciencias Básicas
Departamento	Ciencias de la Computación
Título al que opta	Maestro en Ciencias de la Computación
Nivel:	Especialidad [ ] Maestría [X] Doctorado [ ]

3) Tesis

Título de la Tesis	DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE RIEGO INTELIGENTE BASADO EN INTERNET DE LAS COSAS PARA CULTIVOS PROTEGIDOS EN AGUASCALIENTES, MÉXICO.
Temas/materias	Ingeniería de Software
Nombre del Tutor	Dr. Julio César Ponce Gallegos

Restricciones de Publicación

Por medio de este conducto se autoriza al Departamento de Información Bibliográfica de la Universidad Autónoma de Aguascalientes la publicación electrónica de esta tesis

Sí se autoriza [X] No se autoriza [ ] Se autoriza después de_____años

4) Especificaciones para entrega de tesis en formato electrónico

De acuerdo a la Normatividad: DO-SEE-IT-05 Manual para la elaboración del Trabajo recepcional en los programas de posgrado: Tesis o trabajo práctico.

Integrar en un sólo archivo el trabajo completo en formato PDF.
El texto en formato digital deberá ser identico al de la versión impresa incluyendo la paginación.

Los anexos que contengan archivos electrónicos como multimedia, software, programas autoejecutables o aplicaciones deberán ser agregados en el disco.

Se entregará en un disco óptico no reescribible (CD+/-R o DVD+/-R), deberá de ser cerrado, no estar protegido, etiquetado o rotulado y deberá contar con una caja con portada coincidente con los datos de la portada del trabajo. Excepto cuando, por alguna contingencia, la Dirección General de Investigación y Posgrado, autorice a enviar solo el archivo electrónico.

Método de entrega CD-ROM [ ] DVD [ ] Archivo electrónico [X]

William Alejandro Angulo Martínez
Nombre y firma del Tesista

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mi tutor Dr. Julio César Ponce Gallegos, por su guía, compromiso y paciencia durante el desarrollo de este trabajo. Su orientación académica y su confianza en mis capacidades fueron fundamentales para alcanzar los objetivos planteados.

A mis asesores Dr. Alejandro Padilla Díaz y Dr. Jaime Muñoz Arteaga, por compartir su experiencia, por sus valiosas observaciones y por contribuir al crecimiento académico y técnico de esta investigación. Su acompañamiento constante hizo posible convertir una idea en un proyecto tangible y significativo.

A mi familia, por su apoyo incondicional, por su comprensión en los momentos de ausencia y por motivarme siempre a dar lo mejor de mí. Cada palabra de aliento y cada gesto de cariño fueron el impulso que me permitió seguir adelante.

A quienes alguna vez compartieron conmigo sueños, retos o silencios, y que por distintos motivos ya no están presentes: gracias por su granito de arena.

A todos ellos, gracias por formar parte de este logro, que no solo representa el cierre de una etapa académica, sino también el resultado del esfuerzo compartido y del apoyo de quienes siempre creyeron en mí.

A mi esposa, mi compañera de vida y de sueños.

Gracias por tu amor, por tu paciencia infinita y por creer en mí incluso en los momentos más difíciles. Tu apoyo constante, tus palabras de aliento y tu comprensión fueron mi mayor impulso para no rendirme. Esta meta es también tuya, porque sin ti este logro no habría sido posible.

A mi madre, por ser mi ejemplo más grande de esfuerzo, amor y perseverancia.

Por enseñarme desde pequeño el valor del trabajo honesto, la dedicación y la fe en lo que uno hace. Gracias por estar siempre, por tu cariño incondicional y por ser la raíz de todo lo que soy.

Con todo mi amor y gratitud, dedico este trabajo a ustedes.

Índice General

Índice de Imágenes

Índice de Diagramas

Índice de Tablas

Índice de Códigos

Resumen

Aguascalientes, ubicado en una región semiárida de México, enfrenta una crisis hídrica constante debido a la escasez de lluvias, la sobreexplotación de acuíferos y la contaminación del agua. La agricultura, siendo una de sus principales actividades económicas, se ve especialmente afectada por estas condiciones. Ante este panorama, los cultivos protegidos representan una alternativa más eficiente en el uso de suelo y agua.

Esta tesis presenta el desarrollo y validación experimental en laboratorio de un sistema de riego inteligente basado en tecnologías de Internet de las Cosas (IoT) y aprendizaje automático (machine learning). El sistema utiliza sensores conectados a una Raspberry Pi para recolectar datos ambientales en tiempo real (como humedad del suelo, temperatura y luz), los cuales son enviados vía MQTT a una aplicación web que actúa como sistema central de monitoreo y control. A partir del análisis de estos datos mediante un modelo de machine learning, el sistema demuestra su capacidad para tomar decisiones automáticas simuladas orientadas a optimizar el riego.

El objetivo principal es contribuir al uso eficiente del agua en cultivos protegidos, alineándose con los objetivos de sostenibilidad del estado de Aguascalientes y de México a nivel nacional. La investigación busca no solo ofrecer una solución técnica viable, sino también aportar al desarrollo de sistemas agrícolas resilientes y sostenibles ante los desafíos del cambio climático y el crecimiento urbano.

Palabras clave: riego inteligente, internet de las cosas, aprendizaje automático, agricultura de precisión, cultivos protegidos, sostenibilidad hídrica

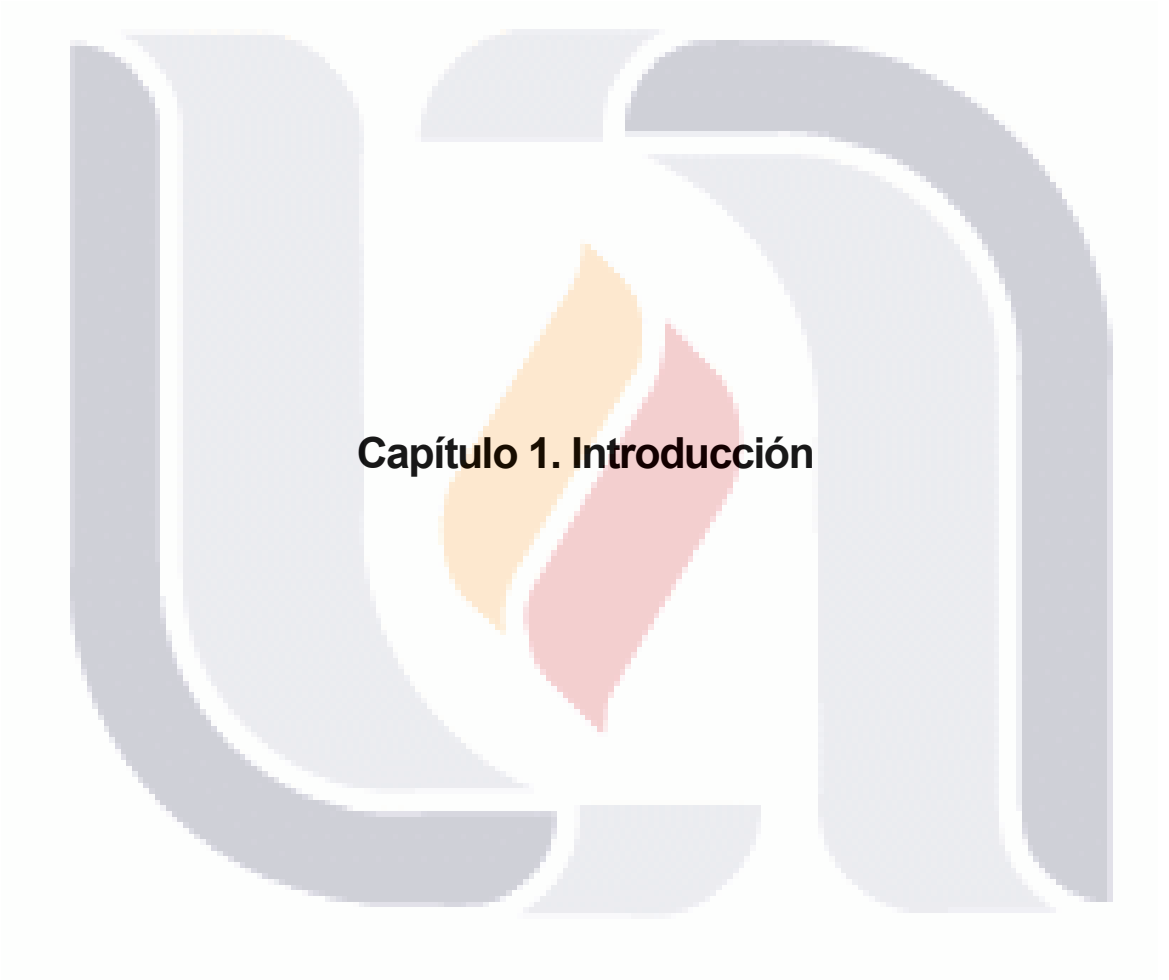
Abstract

Aguascalientes, located in a semi-arid region of Mexico, faces a constant water crisis due to low rainfall, overexploitation of aquifers, and water pollution. Agriculture, being one of its main economic activities, is particularly affected by these conditions. In this context, protected crops represent a more efficient alternative for land and water use.

This thesis presents the development and laboratory validation of an intelligent irrigation system based on Internet of Things (IoT) and machine learning technologies. The system uses sensors connected to a Raspberry Pi to collect real-time environmental data (such as soil moisture, temperature, and light), which are transmitted via MQTT to a web application that serves as the central monitoring and control system. Through the analysis of these data using a machine learning model, the system demonstrates its ability to perform simulated automated decision-making aimed at optimizing irrigation.

The main objective is to contribute to the efficient use of water in protected crops, in alignment with the sustainability goals of the state of Aguascalientes and of Mexico at the national level. The research seeks not only to offer a viable technical solution but also to contribute to the development of resilient and sustainable agricultural systems in the face of climate change and urban growth challenges.

Keywords: smart irrigation, internet of things, machine learning, precision agriculture, protected crops, water sustainability



Capítulo 1. Introducción

1. Introducción

1.1. Contexto

Aguascalientes, un estado situado en el centro de México, se encuentra en una región caracterizada por su clima semiárido, lo que representa un reto significativo para la gestión de recursos hídricos y la agricultura. La escasez de agua se ha convertido en una problemática crónica para el estado, exacerbada por factores como largos períodos de sequía, precipitaciones insuficientes y un aumento en la demanda de agua provocado por el crecimiento de la población y la expansión urbana [1]. Esta situación se agrava aún más debido a la sobreexplotación de los acuíferos y la contaminación del agua, poniendo en riesgo el suministro de agua potable y la sostenibilidad de las actividades agrícolas, vitales para la economía local [1].

La agricultura juega un papel crucial en el sustento económico y social de Aguascalientes. Frente a las adversidades climáticas y los limitados recursos hídricos, los agricultores han buscado alternativas para continuar produciendo de manera eficiente. Los cultivos protegidos, tales como los que se desarrollan en invernaderos y otras estructuras similares, surgen como una solución innovadora frente a estas adversidades. Estos sistemas permiten un control más preciso del ambiente de cultivo, lo que se traduce en rendimientos más altos y un uso más eficiente del agua y del espacio, elementos críticos en un contexto de escasez hídrica y presión sobre las tierras agrícolas.

En respuesta a estos desafíos, el desarrollo e implementación de tecnologías avanzadas, como los sistemas de riego inteligente basados en la tecnología de Internet de las Cosas (IoT), representan una vía prometedora para mejorar la gestión del agua en la agricultura [2]. Mediante la utilización de sensores y dispositivos conectados, estos sistemas pueden monitorear en tiempo real variables clave como la humedad del suelo, la temperatura y la luminosidad, permitiendo ajustes precisos y automáticos en el riego. Este enfoque no solo busca optimizar el uso del agua, sino también alinear las prácticas agrícolas con estrategias de

desarrollo sostenible, contribuyendo a los esfuerzos locales y nacionales para enfrentar la escasez hídrica y fomentar una agricultura resiliente al cambio climático [3], [4].

Esta investigación se inscribe en un momento crítico para Aguascalientes, buscando aportar soluciones concretas a los retos de gestión del agua que enfrenta el estado, al tiempo que se alinea con los objetivos de desarrollo sostenible y de mitigación de los efectos del crecimiento urbano [3], [4]. Al hacerlo, no solo aborda una necesidad inmediata, sino que también se proyecta hacia la construcción de un futuro más sostenible y próspero para la agricultura en Aguascalientes.

1.2. Problema de Investigación

En Aguascalientes, México, la escasez de agua y las condiciones climáticas adversas imponen desafíos significativos al sector agrícola, un componente vital de la economía local. La limitada disponibilidad de agua y los métodos de riego tradicionales, que a menudo resultan ineficientes, ponen en riesgo la sostenibilidad a largo plazo de la agricultura en la región. Estos métodos no solo contribuyen a la sobreexplotación de los acuíferos [1], sino que también aumentan la vulnerabilidad de los cultivos ante condiciones climáticas extremas, afectando la productividad agrícola y la seguridad alimentaria. Además, la contaminación del agua agrava la situación, limitando aún más los recursos hídricos disponibles para uso agrícola [1].

Frente a este escenario, emerge la necesidad imperiosa de explorar y adoptar soluciones tecnológicas innovadoras que permitan una gestión más eficiente del agua. La tecnología de IoT presenta una oportunidad prometedora para revolucionar los sistemas de riego mediante el monitoreo y control en tiempo real del uso del agua, adaptándose precisamente a las necesidades de los cultivos [5]. Sin embargo, a pesar del potencial evidente de esta tecnología para mejorar la eficiencia en el uso del agua, su implementación en la agricultura de Aguascalientes aún es incipiente y enfrenta varios obstáculos, desde la falta de conocimiento y recursos, hasta la resistencia al cambio por parte de los agricultores.

Este estudio se propone abordar esta brecha mediante el desarrollo y validación experimental de un sistema de riego inteligente que optimice el uso del agua y que, a futuro, pueda implementarse de manera accesible y práctica para los agricultores. A través de este enfoque, se busca contribuir a la transformación de la agricultura en Aguascalientes hacia prácticas más sostenibles y resilientes, alineando la innovación tecnológica con los objetivos de desarrollo sostenible del estado y la nación.

Por lo tanto, el problema central de esta investigación se articula en torno a cómo la tecnología de IoT puede ser aprovechada para desarrollar un sistema de riego inteligente que responda a las condiciones específicas de Aguascalientes, mejorando la gestión del agua en la agricultura y contribuyendo a la sostenibilidad y eficiencia del sector ante los retos impuestos por la escasez hídrica y el cambio climático.

1.3. Formulación de Objetivos

1.3.1. Objetivo General

Desarrollar y validar experimentalmente un sistema de riego inteligente basado en IoT que permita optimizar el consumo de agua en cultivos protegidos en Aguascalientes, México, mediante la recolección y análisis de datos ambientales, la predicción del nivel de humedad del suelo y el control automatizado del riego.

1.3.2. Objetivos Específicos

1. **Integrar tecnología de IoT en el sistema de riego para monitorear variables críticas del entorno.**

   Incorporar sensores y dispositivos IoT para monitorear variables críticas como la humedad del suelo, temperatura del aire y del suelo, niveles de luz, y concentraciones de CO2. Estos datos serán fundamentales para probar ajustes automáticos simulados de los ciclos de riego y optimizar el uso del agua.

2. **Diseñar y desarrollar una aplicación web que facilite la validación y manejo de un sistema de riego inteligente.**
   Esta herramienta permitirá el acceso a información en tiempo real y el control preciso del riego, basado en datos específicos de los sensores.

3. **Incorporar un modelo predictivo de machine learning que sugiera el momento óptimo para activar el riego.**
   Este modelo analizará los datos históricos para anticipar caídas en la humedad del suelo y así optimizar el uso del agua.

4. **Validar el sistema en un entorno de pruebas controlado.**
   Se simularán las condiciones operativas del sistema para evaluar su desempeño, identificar posibles errores y ajustar su funcionamiento antes de una implementación real.

5. **Documentar detalladamente el proceso de diseño, desarrollo y validación del sistema.**
   La documentación incluirá diagramas, flujos de trabajo, código fuente relevante, resultados de simulaciones y análisis de viabilidad para futuras implementaciones en campo.

1.4. Justificación

La realización de este proyecto se justifica desde varias perspectivas, destacando su relevancia, valor científico y practicidad para abordar los desafíos críticos de gestión del agua en Aguascalientes, México. Estos elementos subrayan la importancia de desarrollar y evaluar experimentalmente un sistema de riego inteligente basado en IoT para los cultivos protegidos en la región.

1.4.1. Pertinencia

La escasez de agua es un desafío persistente en Aguascalientes, exacerbado por condiciones climáticas extremas como largos periodos de sequía y precipitaciones insuficientes [1]. La agricultura, siendo una de las principales actividades económicas del estado, se ve particularmente afectada por estas condiciones adversas. El desarrollo y validación de un sistema de riego inteligente orientado a optimizar el uso del agua representa una solución crucial y oportuna. Este proyecto no solo aborda un problema ambiental y económico significativo, sino que también se alinea con los objetivos estratégicos del Plan de Desarrollo del Estado 2022-2027 de Aguascalientes [3] y el Plan Nacional de Desarrollo 2025-2030 de México [4], lo que resalta su relevancia a nivel local y nacional.

1.4.2. Valor Científico

La validación experimental de un sistema de riego inteligente contribuye al avance del campo de la agricultura de precisión, un área de investigación que explora cómo las tecnologías avanzadas pueden mejorar la eficiencia y sostenibilidad de la agricultura [2]. Al recopilar y analizar datos en tiempo real sobre variables críticas como la humedad del suelo, la temperatura y la luz solar, este proyecto genera información valiosa sobre la optimización del uso del agua en la agricultura. Este conocimiento no solo tiene el potencial de mejorar las prácticas de riego en Aguascalientes sino también de proporcionar un modelo replicable para otras regiones con desafíos hídricos similares.

1.4.3. Valor Práctico

Desde una perspectiva práctica, el sistema propuesto busca servir como una referencia tecnológica aplicable a las condiciones de la agricultura en Aguascalientes. Al promover una gestión más eficiente del agua, pretende ofrecer una herramienta capaz de reducir el consumo hídrico, aumentar la productividad de los cultivos y fortalecer la sostenibilidad de las prácticas agrícolas. La aplicación web desarrollada como parte del proyecto se concibe como una plataforma que facilitará el monitoreo y control del riego de manera centralizada y accesible,

permitiendo que los agricultores puedan adaptar el sistema a diferentes escenarios productivos [5]. Este enfoque plantea una alternativa práctica y escalable que puede contribuir al fortalecimiento de la sostenibilidad agrícola y a la modernización del manejo del agua en la región. A largo plazo, se espera que los resultados del proyecto sienten las bases para futuras implementaciones en campo, promoviendo la adopción de tecnologías IoT en la agricultura y alineándose con los objetivos regionales y nacionales de conservación del agua y mitigación del cambio climático.

En conjunto, la justificación de este proyecto radica en su capacidad para enfrentar de manera efectiva una problemática ambiental crítica, contribuir al avance científico en la agricultura de precisión [2], y ofrecer una base tecnológica sólida para futuras aplicaciones prácticas en la agricultura de Aguascalientes. Este enfoque integral asegura que el proyecto tenga un impacto significativo tanto en el ámbito académico como en el práctico, promoviendo el uso eficiente de los recursos hídricos en la agricultura y apoyando los objetivos de desarrollo sostenible de la región.

1.5. Alcance y Limitaciones

1.5.1. Alcance

El presente estudio tiene como propósito el desarrollo y validación funcional de un prototipo de sistema de riego inteligente basado en IoT, enfocado en su aplicación futura en cultivos protegidos en la región de Aguascalientes, México. El objetivo principal consiste en contribuir a la optimización del uso del agua en procesos agrícolas mediante la recolección, procesamiento y análisis de datos ambientales y de humedad del suelo en tiempo real, así como la simulación del control automatizado del riego en función de dicha información.

El sistema desarrollado contempla una arquitectura distribuida y modular que incluye los siguientes elementos:

- Un conjunto de sensores físicos conectados a una Raspberry Pi, que actúa como nodo de adquisición de datos y registra variables como humedad del

suelo, humedad relativa ambiental, temperatura del suelo, temperatura del aire, niveles de luz y concentración de CO2.

- El uso del protocolo MQTT (Message Queuing Telemetry Transport) para la transmisión eficiente de los datos sensados hacia un bróker HiveMQ, permitiendo la comunicación entre el hardware y el sistema central.

- Un backend desarrollado en Django con Django REST Framework (DRF), que recibe y procesa los datos provenientes del bróker MQTT. Este backend almacena la información en una base de datos especializada para series temporales (TimescaleDB) y habilita la visualización y control del sistema mediante el uso de WebSockets y una API RESTful.

- Un frontend desarrollado en Vue.js con el framework de diseño Vuetify, que permite la interacción con el sistema en un entorno de validación, incluyendo la visualización gráfica de datos históricos y en tiempo real, así como el control manual o automatizado simulado del riego.

- La integración de un modelo de aprendizaje automático (machine learning) que tiene como finalidad realizar predicciones sobre los niveles futuros de humedad del suelo y sugerir automáticamente el momento óptimo para activar el riego.

- Un sistema de control remoto para el encendido de una bomba de agua, simulado mediante la misma red MQTT desde el backend hacia la Raspberry Pi.

El diseño del sistema se plantea con capacidad de escalabilidad, de manera que pueda adaptarse a diferentes tipos de cultivos protegidos y extenderse para monitorear nuevas variables, controlar múltiples nodos o integrarse con plataformas agrícolas existentes. Asimismo, se prioriza la modularidad del software para facilitar

futuras actualizaciones, mejoras en la precisión del modelo predictivo y ajustes según condiciones geográficas específicas.

1.5.2. Limitaciones

A pesar del alcance técnico del sistema y la solidez de su arquitectura, el desarrollo del proyecto presenta ciertas limitaciones que condicionan su validación en un entorno agrícola operativo. Estas limitaciones son principalmente de tipo logístico, temporal y económico, y se detallan a continuación:

1. **Ausencia de validación en campo real:** Por cuestiones de tiempo y recursos, no se contempla en esta etapa la instalación del sistema en un cultivo protegido operativo. En consecuencia, todas las pruebas funcionales se prevén en entornos controlados, sin exposición directa a las variaciones de un ambiente agrícola real.

2. **Uso de datos sintéticos para el modelo predictivo:** Debido a la falta de bases de datos locales con series temporales de humedad del suelo, el modelo de machine learning se entrena con información generada artificialmente. Esto limita su capacidad de generalización en escenarios reales, por lo que se proyecta su reentrenamiento con datos obtenidos de futuras pruebas en campo.

3. **Restricciones en el hardware disponible:** El prototipo se diseña con una configuración mínima de sensores y componentes electrónicos, lo cual restringe la evaluación de aspectos como durabilidad, resistencia ambiental o tolerancia a fallos, que deberán comprobarse en etapas posteriores.

4. **Imposibilidad de medir el impacto real en el ahorro de agua:** Dado que no se dispone aún de datos empíricos provenientes de un entorno agrícola real, el impacto ambiental y económico se estima de forma teórica o mediante comparaciones con antecedentes documentados.

Estas limitaciones delimitan el alcance del trabajo al nivel de prototipo funcional validado en condiciones controladas, sin comprometer la validez técnica ni la viabilidad del sistema. Se proyecta que, en futuras etapas, el sistema pueda instalarse y probarse en un entorno real de cultivo, permitiendo validar plenamente su efectividad, confiabilidad y contribución al uso sostenible del recurso hídrico en la agricultura protegida.

Capítulo 2. Marco Teórico

2. Marco Teórico

El desarrollo y aplicación de sistemas de riego inteligentes basados en IoT representa un avance significativo en la optimización del uso del agua y la mejora de la productividad agrícola en condiciones de escasez hídrica. Este capítulo establece el marco teórico necesario para comprender los fundamentos y las implicaciones de la adopción de tecnologías IoT en la agricultura, particularmente en el contexto de cultivos protegidos en Aguascalientes, México. A través de una revisión exhaustiva de la literatura, se abordan los conceptos clave relacionados con la gestión sostenible del agua, el crecimiento urbano y su interacción con la agricultura moderna, con énfasis en las soluciones tecnológicas que buscan hacer frente a estos desafíos.

El capítulo se estructura a partir de una descripción de los principios básicos del IoT y su relevancia en el desarrollo de sistemas de riego automatizados y adaptativos, capaces de responder de manera eficiente a las necesidades hídricas específicas de los cultivos. Luego, se analiza la gestión sostenible del agua como un componente esencial de la agricultura en regiones semiáridas, identificando prácticas y tecnologías que contribuyen a su conservación y uso eficiente. Además, se examina el impacto del crecimiento urbano en la disponibilidad de recursos hídricos y la función de los sistemas inteligentes de riego como medida de mitigación, promoviendo una agricultura más sostenible y resiliente.

El análisis incorpora estudios de caso y trabajos previos relevantes que demuestran la aplicación de tecnologías IoT en la agricultura, ofreciendo una visión general de los avances tecnológicos y de sus beneficios potenciales. Este marco teórico proporciona el sustento conceptual de la presente investigación y establece una base sólida para el desarrollo de soluciones inteligentes en la gestión del agua dentro de la agricultura de precisión.

En síntesis, se destaca la importancia de la integración de tecnologías avanzadas en la agricultura como estrategia para enfrentar los retos actuales y futuros asociados a la gestión del agua y la sostenibilidad agrícola.

2.1. Sistemas de Riego

El riego constituye una de las prácticas agrícolas más relevantes para garantizar la disponibilidad de agua en las etapas críticas del desarrollo de los cultivos. Los sistemas de riego se definen como el conjunto de infraestructuras, dispositivos y procedimientos diseñados para distribuir agua de manera controlada sobre una superficie cultivada, con el propósito de satisfacer las necesidades hídricas de las plantas y mantener su equilibrio fisiológico [6].

En esencia, un sistema de riego busca compensar la deficiencia de precipitación o irregularidad en la disponibilidad del agua, asegurando una producción agrícola constante y sostenible.

A lo largo de la historia, los sistemas de riego han evolucionado desde métodos tradicionales basados en el flujo por gravedad hasta soluciones modernas controladas electrónicamente. En términos generales, se reconocen tres tipos principales de sistemas:

1. Riego por superficie: se basa en la distribución del agua mediante canales o surcos, aprovechando la gravedad. Aunque es un método simple y de bajo costo, presenta una eficiencia relativamente baja debido a las pérdidas por escurrimiento y evaporación [7].

2. Riego por aspersión: utiliza presión hidráulica para proyectar el agua en forma de lluvia artificial sobre los cultivos. Este sistema permite una distribución más uniforme, pero requiere un mayor consumo energético y mantenimiento de los emisores.

3. Riego por goteo o microaspersión: suministra el agua directamente a la zona radicular de las plantas mediante emisores localizados. Se considera uno de los métodos más eficientes, con eficiencias de aplicación que pueden superar el 90 %, reduciendo el desperdicio de agua y fertilizantes [8].

En las últimas décadas, la integración de tecnologías digitales, sensores y sistemas de automatización ha impulsado el desarrollo de sistemas de riego inteligentes, capaces de ajustar el caudal, la frecuencia y la duración del riego según las condiciones reales del cultivo y del entorno [9]. Estos sistemas utilizan información proveniente de sensores de humedad del suelo, temperatura, radiación solar y condiciones atmosféricas, para tomar decisiones automáticas que optimicen el uso del recurso hídrico.

Particularmente en regiones semiáridas como Aguascalientes, donde la disponibilidad de agua es limitada, los sistemas de riego inteligentes representan una alternativa tecnológica para aumentar la eficiencia del riego, reducir el consumo hídrico y mejorar la sostenibilidad agrícola [10]. Su adopción contribuye al cumplimiento de los Objetivos de Desarrollo Sostenible (ODS), especialmente en lo relativo a la gestión responsable del agua y la producción agrícola sostenible.

2.2. Internet de las Cosas (IoT)

El Internet de las Cosas (IoT, por sus siglas en inglés Internet of Things) se refiere a una red de dispositivos físicos interconectados que recopilan, procesan y transmiten datos a través de internet, permitiendo la automatización y el control remoto de procesos. Según la definición de la International Telecommunication Union [11], la IoT es “una infraestructura global que conecta objetos físicos y virtuales mediante capacidades de identificación, captura de datos, procesamiento y comunicación, facilitando la interacción entre ellos y con el entorno”.

La IoT combina tecnologías de sensorización, comunicaciones inalámbricas, computación en la nube y análisis de datos para generar ecosistemas inteligentes capaces de operar de manera autónoma o semiautónoma. En el contexto agrícola, esto se traduce en sistemas capaces de monitorear variables ambientales y de cultivo en tiempo real, generando información que puede ser utilizada para optimizar la producción, reducir pérdidas y mejorar la sostenibilidad de los recursos naturales [12].

Los dispositivos IoT agrícolas incluyen sensores de humedad, temperatura, pH, radiación solar o concentración de gases; controladores como microprocesadores o microcontroladores (por ejemplo, Raspberry Pi, Arduino o ESP32); y plataformas digitales que centralizan los datos en la nube. La integración de estos elementos permite construir un sistema ciberfísico en el que los datos capturados en el entorno se transforman en información útil para la toma de decisiones.

En los sistemas de riego inteligente, el IoT desempeña un papel esencial al sincronizar el flujo de información entre los sensores de campo, el servidor y los actuadores, posibilitando la automatización del riego según condiciones ambientales o predicciones generadas mediante modelos de aprendizaje automático [13]. Además, la comunicación basada en protocolos ligeros como MQTT (Message Queuing Telemetry Transport) o CoAP (Constrained Application Protocol) permite el intercambio eficiente de datos entre dispositivos con bajo consumo energético, aspecto clave en entornos agrícolas remotos [14].

En conjunto, el IoT en la agricultura representa una de las aplicaciones más prometedoras dentro de la llamada Agricultura 4.0, al integrar conectividad, inteligencia artificial y computación distribuida para lograr una gestión más precisa, sostenible y resiliente de los recursos agrícolas.

2.3. Fundamentos de IoT aplicados a la agricultura

IoT ha transformado diversos sectores industriales mediante su capacidad para conectar dispositivos a internet, permitiendo la recopilación y el análisis de datos en tiempo real para mejorar la toma de decisiones y la eficiencia operativa. En la agricultura, su aplicación ofrece oportunidades de innovación significativas, especialmente en el ámbito del riego inteligente y la gestión de recursos hídricos. Este apartado describe los fundamentos de IoT aplicados a la agricultura, analizando cómo esta tecnología contribuye a la optimización del uso del agua y al aumento de la productividad de los cultivos.

2.3.1. Conectividad y Monitoreo en Tiempo Real

IoT se basa en sensores y dispositivos conectados que recopilan datos sobre condiciones ambientales clave, como la humedad del suelo, la temperatura, la luminosidad y otros factores determinantes para el crecimiento de los cultivos. Estos datos se transmiten a plataformas centralizadas donde pueden ser monitoreados y analizados en tiempo real, proporcionando a los agricultores información precisa para decidir cuándo y cuánto regar, reduciendo el desperdicio de agua y garantizando un manejo más racional de los recursos hídricos [15].

2.3.2. Automatización del Riego

Uno de los beneficios más destacados de IoT en la agricultura es la posibilidad de automatizar el riego mediante el uso de algoritmos y modelos predictivos. Los sistemas inteligentes ajustan los horarios y volúmenes de riego en función de los datos recolectados por los sensores, lo que incrementa la eficiencia del uso del agua y reduce la carga de trabajo manual, optimizando la gestión del tiempo y los recursos [16].

2.3.3. Agricultura de Precisión

IoT constituye un pilar esencial de la agricultura de precisión, una práctica que busca ajustar las estrategias de cultivo a las condiciones específicas de cada parcela. La integración de datos en tiempo real sobre el clima, el suelo y las plantas permite aplicar tratamientos diferenciados, mejorar la salud de los cultivos y maximizar el rendimiento productivo mediante un uso más racional de los recursos [17].

2.3.4. Sostenibilidad y Gestión de Recursos

La incorporación de IoT en la agricultura contribuye a la sostenibilidad mediante un manejo más eficiente del agua y los insumos agrícolas. Al optimizar el riego y reducir la dependencia de fertilizantes o agroquímicos, los sistemas inteligentes favorecen la conservación de los recursos naturales y la reducción del impacto ambiental de las prácticas agrícolas [18].

2.4. Gestión Sostenible del Agua

La gestión sostenible del agua en la agricultura constituye un componente esencial para garantizar la seguridad alimentaria, preservar el medio ambiente y promover el desarrollo socioeconómico, especialmente en regiones áridas y semiáridas como Aguascalientes, México. Esta sección describe las prácticas, estrategias y tecnologías orientadas al uso eficiente y responsable de los recursos hídricos en la agricultura, destacando la relevancia de adoptar enfoques integrales y sostenibles para enfrentar la escasez de agua y los efectos asociados al cambio climático.

2.4.1. Principios de la Gestión Sostenible del Agua

La gestión sostenible del agua se fundamenta en la búsqueda de equilibrio entre las necesidades humanas, la productividad agrícola y la conservación de los ecosistemas naturales. Entre sus principios destacan la eficiencia en el uso del agua, la reducción de la contaminación, la protección de los ciclos hidrológicos y la equidad en el acceso al recurso. En el ámbito agrícola, estos principios se traducen en la adopción de prácticas que minimizan el desperdicio, mejoran la infiltración y retención del agua en el suelo, y garantizan su aprovechamiento óptimo tanto para los cultivos como para el entorno [19].

2.4.2. Tecnologías y Prácticas para la Sostenibilidad Hídrica

Diversas tecnologías y prácticas contribuyen al manejo racional y sostenible del agua en la agricultura moderna:

- **Riego de precisión:** Sistemas como el riego por goteo o la aspersión controlada dirigen el agua específicamente a las zonas de mayor demanda hídrica, reduciendo pérdidas por evaporación y escurrimiento.
- **Sensores de humedad del suelo y sistemas automatizados:** Equipos IoT permiten registrar en tiempo real las condiciones del suelo y ajustar los parámetros de riego de manera dinámica, optimizando el uso del agua sin comprometer el desarrollo de los cultivos.

- **Cultivos resistentes a la sequía:** El desarrollo y la selección de variedades que requieren menor cantidad de agua o toleran periodos prolongados de sequía reducen la presión sobre las fuentes hídricas.
- **Manejo integrado de recursos hídricos:** Estrategias que consideran el uso combinado de aguas superficiales, subterráneas y no convencionales, promoviendo su gestión coordinada entre distintos sectores productivos [18].

2.4.3. Impacto en la Agricultura y el Medio Ambiente

La adopción de prácticas de gestión sostenible del agua contribuye simultáneamente al aumento de la productividad agrícola y a la conservación ambiental. Estas estrategias favorecen la preservación de ecosistemas acuáticos, reducen procesos de salinización y degradación del suelo, y minimizan la contaminación derivada de escorrentías agrícolas. A largo plazo, fortalecen la resiliencia de las comunidades rurales frente a la variabilidad climática y la disminución de la disponibilidad de agua [20].

2.4.4. Desafíos y Oportunidades

La transición hacia modelos de gestión hídrica sostenibles implica retos estructurales y operativos, como la necesidad de inversión tecnológica, la capacitación continua de los productores y la adecuación de marcos normativos que promuevan el uso responsable del agua. No obstante, estos desafíos abren también oportunidades para impulsar la innovación, fortalecer la cooperación interinstitucional y consolidar una agricultura resiliente, eficiente y ambientalmente responsable [21].

2.5. Impacto del Crecimiento Urbano

El crecimiento urbano, entendido como la expansión de las ciudades y el incremento de la población en áreas urbanizadas, constituye uno de los principales factores de presión sobre los recursos naturales, especialmente el agua. Esta sección analiza la forma en que la expansión urbana influye en la disponibilidad, calidad y gestión

del agua destinada a la agricultura, con especial atención a las condiciones de las regiones semiáridas como Aguascalientes, México.

2.5.1. Presión sobre los Recursos Hídricos

A medida que las ciudades crecen, la demanda de agua para fines domésticos, industriales y recreativos aumenta de manera proporcional, generando competencia directa con las necesidades del sector agrícola. Este incremento sostenido favorece la sobreexplotación de ríos y acuíferos, reduciendo la cantidad de agua disponible para el riego y comprometiendo la sostenibilidad de los sistemas agrícolas. En contextos como el de Aguascalientes, donde la escasez hídrica ya representa un problema estructural, el crecimiento urbano intensifica la competencia por recursos limitados, poniendo en riesgo la viabilidad productiva y ambiental de la agricultura regional [22].

2.5.2. Cambios en el Uso del Suelo

El proceso de urbanización suele implicar la conversión de terrenos agrícolas en áreas residenciales, comerciales o industriales, lo que reduce la superficie destinada a la producción de alimentos y altera los equilibrios hidrológicos naturales. Este cambio de uso del suelo modifica los patrones de infiltración y recarga de los acuíferos, y al mismo tiempo incrementa la proporción de superficies impermeables, como calles, techos o pavimentos, aumentando la escorrentía superficial y disminuyendo la retención natural de agua en el subsuelo [23].

2.5.3. Contaminación del Agua

El crecimiento urbano también contribuye a la contaminación de los cuerpos de agua a través de descargas residuales sin tratamiento adecuado y del arrastre de contaminantes provenientes de áreas urbanas. Esta contaminación deteriora la calidad del agua disponible para el riego agrícola, con efectos negativos sobre la salud de los cultivos y, por extensión, sobre la seguridad alimentaria. En regiones como Aguascalientes, donde la agricultura depende en gran medida de sistemas de riego, la preservación de la calidad del agua se considera un componente esencial para garantizar la productividad y sostenibilidad agrícola [24].

2.5.4. Estrategias de Mitigación

Frente a los efectos del crecimiento urbano sobre los recursos hídricos, la adopción de estrategias integradas de mitigación resulta indispensable. Entre las más relevantes se encuentran:

- **Planificación urbana y agrícola coordinada:** Promueve una gestión conjunta del territorio para proteger las zonas agrícolas y los acuíferos estratégicos.
- **Tecnologías urbanas de eficiencia hídrica:** Fomenta la instalación de sistemas de ahorro y reciclaje de agua en las ciudades, reduciendo la demanda sobre las fuentes naturales.
- **Tratamiento y reutilización de aguas residuales:** La inversión en infraestructura de saneamiento permite reutilizar el agua tratada en riego agrícola, disminuyendo la presión sobre los recursos hídricos convencionales.
- **Educación y sensibilización social:** Difunde la importancia del uso racional del agua entre la población urbana y rural, promoviendo prácticas de conservación sostenibles.

La gestión equilibrada del crecimiento urbano y de la disponibilidad de agua para la agricultura requiere un enfoque sistémico que considere simultáneamente las necesidades urbanas y agrícolas. A través de estrategias proactivas y sostenibles, es posible alcanzar un equilibrio entre desarrollo urbano, seguridad alimentaria y conservación hídrica, elementos fundamentales para el desarrollo sostenible en regiones semiáridas como Aguascalientes [25].

2.6. Industria 4.0 y su Aplicación en la Agricultura

La Industria 4.0, también denominada cuarta revolución industrial, se define por la integración de tecnologías digitales avanzadas, como la inteligencia artificial (IA), IoT, la robótica y el análisis de big data, en los procesos productivos. En el ámbito

agrícola, estas tecnologías están transformando las prácticas tradicionales mediante el desarrollo de sistemas inteligentes capaces de optimizar la eficiencia, reducir costos y fortalecer la sostenibilidad de los cultivos [26].

Una de las áreas más relevantes de la Industria 4.0 es la agricultura de precisión, la cual utiliza información en tiempo real para respaldar la toma de decisiones sobre el manejo de los cultivos. Este enfoque comprende el monitoreo de variables ambientales y fisiológicas, como el clima, la humedad del suelo, los niveles de nutrientes y otros factores críticos que inciden directamente en la productividad agrícola. La incorporación de sensores IoT, drones y sistemas de información geográfica (SIG) posibilita una gestión más precisa y adaptativa de los recursos agrícolas, contribuyendo a una agricultura más eficiente y sostenible [27].

2.6.1. Cultivos Protegidos: Innovación y Sostenibilidad

Los cultivos protegidos, como los invernaderos y túneles de plástico, constituyen una estrategia tecnológica orientada a crear ambientes controlados que protegen a las plantas de condiciones climáticas extremas, plagas y enfermedades. Esta técnica incrementa la calidad y el rendimiento de los cultivos, al mismo tiempo que optimiza el uso de recursos esenciales como el agua y los fertilizantes.

En el contexto de Aguascalientes, México, donde la disponibilidad de agua es limitada, los cultivos protegidos representan una alternativa sostenible que permite mantener altos niveles de productividad agrícola con un consumo hídrico reducido. En este tipo de entornos, la integración de sistemas de riego inteligentes basados en IoT contribuye a la optimización del uso del agua, minimiza las pérdidas por evaporación y mejora la eficiencia general de los procesos agrícolas [28].

2.6.2. Internet de las Cosas (IoT) en la Agricultura

IoT se considera una de las tecnologías clave de la Industria 4.0 debido a su capacidad para conectar dispositivos y sistemas a través de redes digitales, facilitando la recopilación, transmisión y análisis de datos en tiempo real. En la

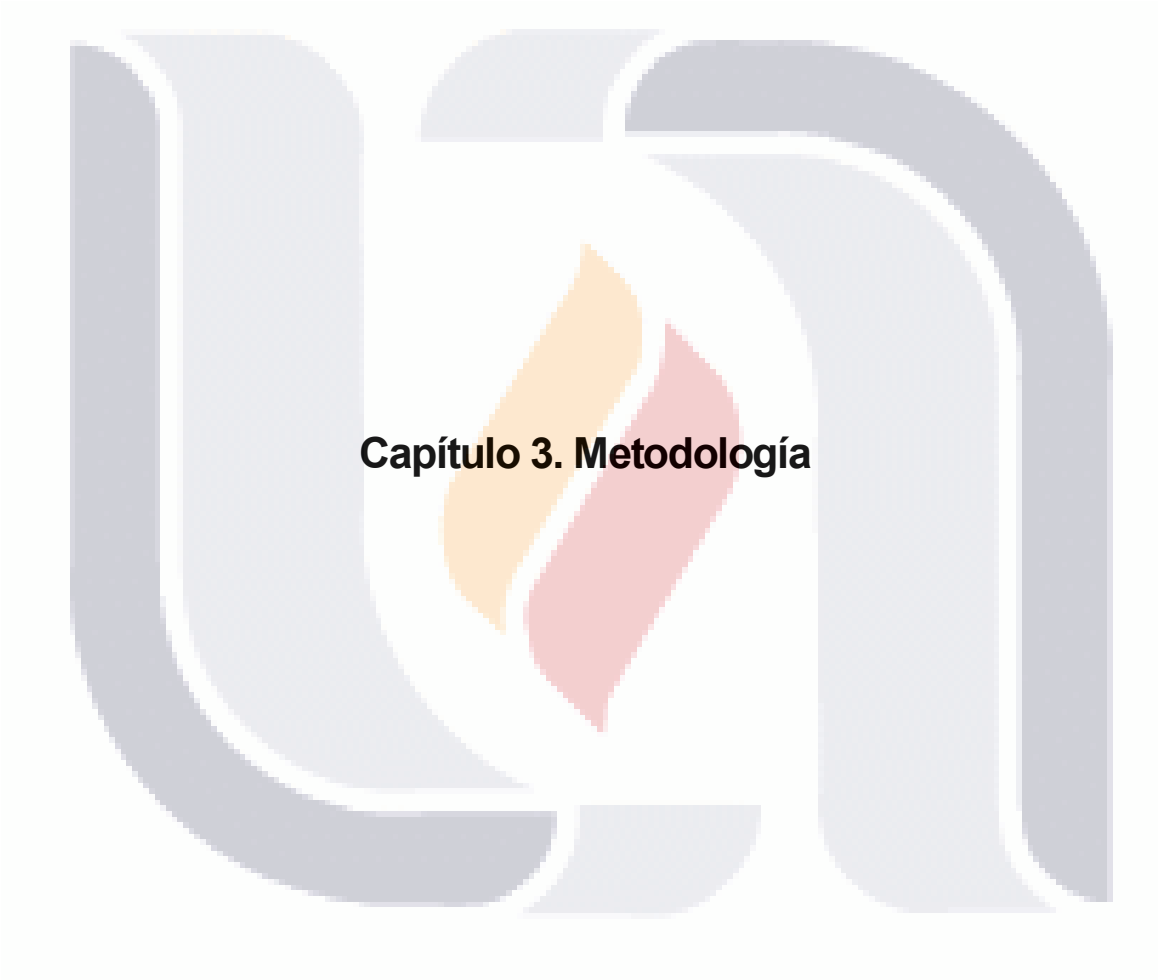
agricultura, su aplicación permite el desarrollo de sistemas de riego inteligentes que ajustan de manera autónoma el suministro de agua conforme a las necesidades hídricas de cada cultivo.

Mediante el uso de sensores que miden humedad del suelo, temperatura, radiación lumínica y concentración de CO_2, los sistemas IoT pueden modificar dinámicamente los patrones de riego, optimizando el uso del agua y promoviendo un crecimiento vegetal saludable. Además, estos sistemas generan alertas ante anomalías, como sequías o exceso de humedad, permitiendo una respuesta temprana y precisa [29].

2.6.3. Justificación del Uso de IoT en Cultivos Protegidos

La combinación de cultivos protegidos e IoT representa una convergencia tecnológica con alto potencial para mejorar la sostenibilidad agrícola en regiones semiáridas como Aguascalientes. Los sistemas de riego inteligentes basados en IoT favorecen el uso racional del agua dentro de los invernaderos, asegurando una irrigación óptima que reduce el desperdicio y maximiza la eficiencia hídrica.

Además, la aplicación de herramientas de análisis de datos y monitoreo en tiempo real fortalece la toma de decisiones agronómicas basadas en evidencia, reduciendo la incertidumbre y mejorando la capacidad de gestión de los productores. Este enfoque se alinea con los objetivos globales de sostenibilidad y seguridad alimentaria, posicionando a Aguascalientes como una región con potencial de liderazgo en la adopción de tecnologías agrícolas innovadoras y resilientes frente al cambio climático [30].



Capítulo 3. Metodología

3. Metodología

El desarrollo de este proyecto se enmarcó en una investigación aplicada de carácter tecnológico, cuyo objetivo fue abordar un problema práctico del sector agrícola mediante el diseño y validación de un sistema informático basado en tecnologías emergentes. El enfoque metodológico adoptado fue cuantitativo, experimental y tecnológico, centrado en la construcción y prueba funcional de un prototipo de sistema de riego inteligente que integró dispositivos IoT, algoritmos de aprendizaje automático y una arquitectura web modular.

La metodología empleada se basó en los principios del modelo de desarrollo iterativo, en el cual se diseñaron, desarrollaron y validaron de forma independiente los distintos componentes del sistema, lo que permitió realizar ajustes progresivos durante el proceso. El enfoque modular facilitó el trabajo paralelo sobre cada subsistema (hardware, backend, frontend, comunicación, predicción y control), garantizando flexibilidad y coherencia en la integración final.

Este capítulo describe las fases metodológicas que guiaron el desarrollo del sistema, abarcando desde el análisis del problema y la definición de requerimientos, hasta el diseño técnico, la construcción del prototipo, la validación funcional en condiciones controladas y la documentación integral del proceso.

3.1. Fase 1: Análisis del problema y definición de requerimientos

Esta fase se enfocó en el estudio de la problemática del uso ineficiente del agua en cultivos protegidos, particularmente en zonas semiáridas como Aguascalientes. Se llevó a cabo una revisión documental de las tecnologías actuales en agricultura inteligente, así como de sistemas de riego automatizado y soluciones IoT aplicadas a entornos de producción controlada.

A partir del análisis, se identificaron los principales factores que inciden en la eficiencia del riego, tales como la humedad del suelo, la temperatura ambiental, la

humedad relativa, el tipo de cultivo y las condiciones del entorno protegido. Esta información permitió establecer los requerimientos técnicos y operativos del sistema, clasificados de la siguiente manera:

Requerimientos funcionales:

- Adquisición en tiempo real de datos ambientales mediante sensores.
- Transmisión eficiente de datos desde los nodos IoT hacia el servidor central.
- Procesamiento de datos y toma de decisiones automatizadas o simuladas.
- Activación remota del sistema de riego mediante comandos controlados.
- Visualización de la información a través de una aplicación web.
- Predicción del nivel de humedad mediante modelos de machine learning.

Requerimientos no funcionales:

- Alta disponibilidad del sistema.
- Bajo consumo energético en los nodos IoT.
- Escalabilidad horizontal para soportar múltiples zonas de riego.
- Interfaz intuitiva y adaptable a distintos dispositivos.
- Integración segura entre módulos mediante protocolos ligeros y confiables.

Los resultados de esta fase sirvieron como base para el diseño técnico y la planificación de las etapas posteriores del proyecto.

3.2. Fase 2: Diseño del Sistema

3.2.1. Diseño de Software

El diseño del software constituyó el eje central del desarrollo del sistema de riego inteligente, al definir la estructura lógica, los componentes tecnológicos y las herramientas que permitieron construir y validar el prototipo funcional. Esta fase se enfocó en establecer una arquitectura robusta, modular y escalable, adecuada para un entorno de experimentación controlada y adaptable a una futura implementación en campo.

- **Lenguajes de programación:** Se seleccionó Python para el desarrollo del backend, debido a su amplia comunidad y a la disponibilidad de librerías como NumPy y Pandas, que facilitaron el manejo y análisis de datos ambientales recopilados por los sensores. Para el frontend se empleó JavaScript junto con el framework Vue.js, con el fin de desarrollar una interfaz interactiva y adaptable que permitiera visualizar la información del sistema y realizar acciones de control durante las pruebas experimentales.

- **Frameworks y plataformas:** Se utilizó Django, un framework de alto nivel para Python, para estructurar el backend del sistema, aprovechando su arquitectura basada en componentes reutilizables y su capacidad para gestionar peticiones web y comunicación con la base de datos de manera segura. En el frontend, Vue.js fue empleado para garantizar una actualización eficiente de los elementos visuales y un manejo fluido del estado de la aplicación.

- **Arquitectura del sistema:** Se adoptó un enfoque modular inspirado en microservicios, lo que permitió el desarrollo, prueba e integración independiente de cada componente del sistema (adquisición de datos, almacenamiento, análisis y visualización). Esta estructura favoreció la escalabilidad y la flexibilidad del prototipo, facilitando la incorporación de nuevos módulos o la modificación de los existentes sin afectar el funcionamiento global del sistema durante la validación experimental.

3.3. Fase 3: Implementación

3.3.1. Desarrollo de Software

La implementación del software se realizó de manera progresiva, siguiendo un enfoque modular que garantizó la funcionalidad y estabilidad del prototipo desarrollado.

- **Conectividad IoT:** Se configuraron protocolos estándar como MQTT, empleado para la comunicación entre los dispositivos conectados y el servidor de procesamiento. Este protocolo se seleccionó por su ligereza y confiabilidad en la entrega de mensajes en tiempo real, lo que permitió simular adecuadamente el flujo de datos entre sensores y plataforma central durante la validación experimental.

- **Monitoreo en tiempo real:** Se desarrolló un tablero de control interactivo que permitió visualizar el comportamiento del sistema en tiempo real. Este tablero integró gráficos dinámicos y representaciones visuales de los datos obtenidos de los sensores, facilitando el análisis y la supervisión de las variables ambientales durante las pruebas.

3.3.2. Integración con Hardware

Para el desarrollo del prototipo se utilizó una Raspberry Pi como controlador central de riego. A continuación, se describen los sensores seleccionados y su integración con el controlador, realizada en un entorno de validación experimental.

Selección de sensores

- Sensor de humedad del suelo: DHT22
  Este sensor permitió medir la humedad y la temperatura del ambiente con alta precisión. El DHT22 se conectó a la Raspberry Pi a través de los pines GPIO mediante comunicación digital.

- Sensor de temperatura del aire y del suelo: DS18B20
  Este sensor digital de temperatura permitió monitorear simultáneamente la temperatura del suelo y del ambiente. Su conexión se realizó mediante el bus de 1-Wire, lo que facilitó la integración de múltiples sensores sobre una misma línea.

- Sensor de niveles de luz: BH1750

El BH1750 se utilizó para medir la intensidad lumínica en lux, conectándose a la Raspberry Pi mediante interfaz I2C. Esta información resultó esencial para analizar la relación entre iluminación y demanda hídrica.

- Sensor de concentraciones de CO_2: SCD30
  El SCD30 permitió medir las concentraciones de dióxido de carbono junto con la humedad y temperatura ambiental, comunicándose con la Raspberry Pi mediante la interfaz I2C. Estos datos complementaron la caracterización del entorno de pruebas.

Proceso de integración

1. Conexión de sensores a la Raspberry Pi
   - Cada sensor se conectó a la Raspberry Pi utilizando sus respectivos protocolos de comunicación, asegurando la estabilidad de las señales mediante resistencias pull-up cuando fue necesario.
   - Se desarrollaron scripts en Python para la lectura de los datos, empleando librerías especializadas como Adafruit_DHT, w1thermsensor, smbus2 y smbus.

2. Configuración del software en la Raspberry Pi
   - Se instaló y configuró el sistema operativo Raspberry Pi OS, habilitando las interfaces necesarias (I2C y 1-Wire).
   - Se desarrolló una aplicación en Python que recopiló, procesó y envió los datos hacia la base de datos central.
   - El sistema incluyó rutinas para simular la activación del riego, de acuerdo con los parámetros definidos por el algoritmo de control.

3. Pruebas y calibración de sensores
   - Se efectuaron pruebas de funcionamiento y calibración conforme a las especificaciones de cada fabricante.

- Se configuraron alertas de supervisión para detectar valores anómalos o inconsistentes durante las mediciones.

Evaluación y optimización

- Se desarrolló un sistema de monitoreo web que permitió visualizar los datos de los sensores en tiempo real, realizar ajustes de parámetros y validar el desempeño del prototipo.
- La integración completa del sistema se probó en condiciones simuladas de cultivo protegido, evaluando su estabilidad, precisión y capacidad de respuesta ante variaciones ambientales.

3.3.3. Descripción de Sensores y Controlador

En esta sección se describieron los sensores seleccionados y la Raspberry Pi utilizada como unidad central de control en el prototipo del sistema de riego inteligente basado en IoT. Cada componente fue elegido por su precisión, confiabilidad y compatibilidad con entornos experimentales de agricultura de precisión.

3.3.3.1. Controlador Central: Raspberry Pi

- **Características:**
  - **Modelo:** Raspberry Pi 4 Model B.
  - **Procesador:** Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz.
  - **Memoria RAM:** 4GB LPDDR4-3200 SDRAM.
  - **Conectividad:** 2.4 GHz y 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE, Gigabit Ethernet.
  - **Puertos:** 2 puertos USB 3.0, 2 puertos USB 2.0, 2 micro HDMI (soporte de hasta 4Kp60), GPIO de 40 pines.
  - **Almacenamiento:** MicroSD.
- **Funcionamiento:**

La Raspberry Pi 4 Model B desempeñó el rol de controlador central del prototipo, gestionando la adquisición y procesamiento de los datos provenientes de los sensores. Su capacidad de cómputo permitió ejecutar los algoritmos de análisis en tiempo real y coordinar las rutinas de control del sistema durante la validación experimental.

- Conexión de sensores: Los sensores se conectaron a la Raspberry Pi mediante los puertos GPIO y las interfaces I2C y 1-Wire. Se implementaron scripts en Python para la lectura, almacenamiento y procesamiento de los datos obtenidos.
- Procesamiento de datos: La Raspberry Pi ejecutó algoritmos de análisis y predicción que interpretaron las mediciones ambientales, simulando decisiones automáticas sobre el riego en función de las variables monitoreadas.
- Simulación de automatización del riego: En lugar de una activación real de válvulas o bombas, se configuró un mecanismo de simulación del control de riego, lo que permitió validar la lógica de funcionamiento y verificar la correcta respuesta del sistema ante distintos escenarios simulados.

3.3.3.2. Sensor de Humedad del Suelo: DHT22

- **Características:**
  - **Tipo:** Sensor digital de temperatura y humedad.
  - **Rango de humedad:** 0-100% RH (Humedad Relativa).
  - **Precisión de humedad:** ±2% RH.
  - **Rango de temperatura:** -40 a 80 °C.
  - **Precisión de temperatura:** ±0.5 °C.
  - **Tiempo de respuesta:** 2 segundos.
  - **Interfaz:** Digital.
- **Funcionamiento:**

  El sensor DHT22 se utilizó para registrar valores de temperatura y humedad del entorno durante la validación experimental del prototipo. Aunque

comúnmente se emplea para medir la humedad relativa del aire, en este proyecto se configuró para estimar los niveles de humedad del suelo dentro de condiciones controladas.

El DHT22 opera mediante un sensor capacitivo que mide la humedad y un termistor que mide la temperatura, enviando la información digitalmente a la Raspberry Pi. Esta comunicación digital eliminó la necesidad de un convertidor analógico-digital adicional, simplificando la integración con el controlador central.

Gracias a su precisión, estabilidad y bajo costo, el DHT22 resultó adecuado para la verificación del desempeño del sistema de adquisición de datos, proporcionando mediciones confiables para el análisis de las variables ambientales dentro del entorno de prueba.

3.3.3.3. Sensor de Temperatura del Aire y del Suelo: DS18B20

- **Características:**
  - **Tipo:** Sensor digital de temperatura.
  - **Rango de temperatura:** -55 a 125 °C.
  - **Precisión:** ±0.5 °C en el rango de -10 a 85 °C.
  - **Interfaz:** 1-Wire.
  - **Resolución:** 9 a 12 bits, programable.
  - **Tiempo de respuesta:** Conversión de temperatura en menos de 750 ms.
- **Funcionamiento:**

  El sensor DS18B20 se empleó para medir la temperatura tanto del aire como del suelo durante las pruebas experimentales del prototipo. Su protocolo de comunicación 1-Wire permitió la conexión de varios sensores en una misma línea, lo que simplificó el cableado y facilitó la expansión del sistema.

  Gracias a su precisión y rango operativo amplio, el DS18B20 proporcionó mediciones confiables de temperatura que fueron utilizadas para analizar la respuesta térmica del sistema y validar el desempeño del algoritmo de control.

Las lecturas obtenidas se transmitieron digitalmente a la Raspberry Pi, donde fueron procesadas en tiempo real por los scripts desarrollados en Python. Este sensor demostró ser adecuado para la etapa de validación experimental, permitiendo evaluar la capacidad del sistema para adaptarse a variaciones térmicas simuladas.

3.3.3.4. Sensor de Niveles de Luz: BH1750

- **Características:**
  - **Tipo:** Sensor digital de luz ambiental.
  - **Rango de medición:** 0-65,535 lux.
  - **Precisión:** Alta precisión en un amplio rango de luminancia.
  - **Interfaz:** I2C.
  - **Consumo de energía:** Bajo consumo de energía.
  - **Tiempo de respuesta:** 16 ms a 120 ms dependiendo de la resolución seleccionada.
- **Funcionamiento:**

  El sensor BH1750 se utilizó para medir la intensidad lumínica durante la validación experimental del prototipo, proporcionando valores expresados en lux. Este dispositivo empleó la interfaz I2C para comunicarse con la Raspberry Pi, lo que permitió una lectura rápida y estable de los datos.

  Las mediciones obtenidas fueron registradas y analizadas en tiempo real, facilitando la evaluación del comportamiento del sistema ante variaciones de luminosidad simuladas. Si bien en una aplicación en campo los datos de este sensor podrían emplearse para ajustar automáticamente los horarios de riego, en esta etapa su función se limitó a verificar la correcta adquisición y transmisión de información lumínica dentro del entorno de prueba.

  Gracias a su amplio rango de medición y alta sensibilidad, el BH1750 resultó adecuado para validar la precisión del sistema de monitoreo ambiental y comprobar la estabilidad de la comunicación entre los distintos módulos del sistema.

3.3.3.5. Sensor de Concentraciones de CO2: SCD30

- **Características:**
  - **Tipo:** Sensor de dióxido de carbono (CO2), humedad y temperatura.
  - **Rango de medición de CO2:** 400-10,000 ppm.
  - **Precisión de CO2:** ±(30 ppm + 3% de la lectura).
  - **Rango de humedad:** 0-100% RH.
  - **Precisión de humedad:** ±2% RH.
  - **Rango de temperatura:** -40 a 70 °C.
  - **Precisión de temperatura:** ±0.4 °C.
  - **Interfaz:** I2C.
  - **Consumo de energía:** Bajo consumo de energía.
- **Funcionamiento:**

  El sensor SCD30 se empleó para medir simultáneamente las concentraciones de dióxido de carbono, la humedad relativa y la temperatura del entorno durante la validación experimental del prototipo. Este dispositivo utilizó la interfaz I2C para comunicarse con la Raspberry Pi, permitiendo la adquisición estable y continua de datos ambientales.

  Las mediciones de CO_2 resultaron especialmente útiles para analizar la relación entre la concentración de gases y las condiciones ambientales simuladas, ya que el dióxido de carbono influye directamente en los procesos de fotosíntesis y transpiración vegetal.

  El conjunto de variables registradas por el SCD30 complementó la información proveniente de otros sensores, posibilitando un monitoreo integral del entorno de pruebas y aportando datos relevantes para la verificación del desempeño del sistema de monitoreo ambiental.

  Gracias a su precisión y estabilidad de lectura, el SCD30 demostró ser un componente adecuado para la validación del modelo de adquisición de datos del sistema de riego inteligente, sirviendo como referencia para futuras aplicaciones en entornos agrícolas reales.

3.3.3.6. Integración y Monitoreo

La integración de los sensores con la Raspberry Pi permitió realizar un monitoreo continuo y preciso de las condiciones ambientales y del suelo dentro del entorno de validación experimental. Los datos obtenidos fueron procesados en tiempo real mediante los scripts desarrollados en Python, lo que permitió verificar el correcto funcionamiento del flujo de adquisición, transmisión y análisis de información.

Durante las pruebas, se evaluó la estabilidad del sistema y la coherencia de los datos registrados, comprobando la capacidad del prototipo para operar de forma autónoma y mantener la comunicación constante entre los diferentes módulos. Si bien no se realizaron ajustes automáticos sobre un sistema de riego real, se validó la lógica de control que permitiría dicha automatización en futuras etapas de implementación.

Esta integración demostró que la arquitectura diseñada es funcional, eficiente y adaptable a distintos escenarios de uso, validando el desempeño general del sistema propuesto para la gestión inteligente del riego. Con esta configuración, el prototipo consolidó su capacidad para optimizar el manejo del agua en condiciones simuladas y sentó las bases para la adopción de prácticas agrícolas más sostenibles y resilientes en aplicaciones futuras.

3.4. Fase 4: Validación en entorno de laboratorio

La validación del sistema se realizó en un entorno controlado con el propósito de comprobar su funcionamiento integral, la consistencia del flujo de datos y la capacidad de respuesta ante diferentes condiciones simuladas de riego. Esta fase representó la culminación del desarrollo del prototipo, enfocándose en evaluar su desempeño bajo condiciones reproducibles y monitoreadas.

Las pruebas efectuadas incluyeron:

- Simulación de datos sensoriales para verificar el flujo completo de información desde los sensores hasta el frontend.

- Pruebas de latencia para medir el tiempo de respuesta entre la adquisición de los datos, su procesamiento en el backend y su visualización en la interfaz web.
- Simulación de la activación remota de la bomba de agua mediante la interfaz web y comandos transmitidos por el protocolo MQTT, validando la comunicación entre los distintos módulos del sistema.
- Evaluación del modelo predictivo de humedad del suelo, verificando la correspondencia entre las sugerencias de riego generadas y los valores esperados según los escenarios establecidos.

Los resultados obtenidos demostraron que el sistema operó correctamente, integrando de manera efectiva los componentes de hardware, software y comunicación. Se confirmó la capacidad del prototipo para procesar datos en tiempo real, generar predicciones coherentes y ejecutar respuestas automáticas o manuales ante las condiciones simuladas.

Esta fase permitió validar el desempeño funcional y la estabilidad del sistema propuesto, consolidando su viabilidad técnica como herramienta de apoyo para la gestión inteligente del riego en entornos controlados y su potencial para futuras aplicaciones agrícolas.

3.5. Fase 5: Documentación del proceso

A lo largo del desarrollo del proyecto se elaboró un registro sistemático y detallado de todas las etapas del trabajo, con el objetivo de garantizar la trazabilidad del proceso y facilitar la replicación o ampliación futura del sistema. La documentación generada constituyó una parte esencial del proyecto, ya que permitió mantener una organización clara entre los componentes técnicos, los resultados obtenidos y las decisiones de diseño adoptadas.

Entre los materiales recopilados se incluyeron:

- Diagramas de arquitectura del sistema y flujos de datos, que describen la interacción entre los módulos de hardware, software y comunicación.
- Esquemas eléctricos y de conexión de sensores, elaborados para representar con precisión la configuración empleada durante la validación experimental.
- Bitácora técnica de incidencias, donde se registraron errores detectados, ajustes realizados y soluciones aplicadas en cada fase del desarrollo.
- Código fuente estructurado por módulos y subcomponentes, documentado mediante comentarios y convenciones estandarizadas para facilitar su comprensión y mantenimiento.
- Capturas de la interfaz de usuario y registros de las pruebas realizadas, que evidencian el comportamiento del sistema y sus funcionalidades principales.
- Análisis de rendimiento, en el que se evaluaron aspectos como la latencia del sistema, el procesamiento de datos y la estabilidad de la comunicación entre módulos.

Esta documentación se integró como parte sustantiva de la tesis, con la finalidad de servir como referencia para futuras aplicaciones o ampliaciones del sistema, así como guía metodológica para investigaciones o desarrollos posteriores en el ámbito de los sistemas de riego inteligente basados en IoT.

Capítulo 4. Desarrollo e Implementación

4. Desarrollo e Implementación

El presente capítulo describe el proceso de desarrollo e implementación del SIRCA-IoT (Sistema Inteligente de Riego y Control Automatizado basado en IoT), propuesto en este estudio. El sistema fue concebido como un prototipo funcional orientado a demostrar la viabilidad técnica y operativa de integrar tecnologías de sensorización, comunicación inalámbrica, procesamiento de datos y aprendizaje automático en el contexto de cultivos protegidos en Aguascalientes, México.

El desarrollo se estructuró bajo una arquitectura modular que permitió abordar de forma independiente los diferentes subsistemas que conforman la solución: el subsistema IoT para la adquisición y transmisión de datos ambientales, la plataforma web encargada del procesamiento, visualización y control del sistema, y el subsistema predictivo basado en técnicas de machine learning. Esta organización favoreció el diseño iterativo y la validación individual de cada componente, garantizando la coherencia en la integración final.

En esta etapa del proyecto, el sistema SIRCA-IoT fue implementado y validado en un entorno controlado que permitió reproducir las condiciones de funcionamiento esperadas en un cultivo protegido, asegurando el monitoreo continuo de variables ambientales relevantes y la automatización del proceso de riego en función de los datos obtenidos y las predicciones generadas. El enfoque experimental adoptado permitió verificar la comunicación entre los dispositivos IoT, la estabilidad del flujo de datos, la eficiencia del backend en el manejo de información en tiempo real y la precisión del modelo predictivo en la estimación del nivel de humedad del suelo.

El desarrollo se organizó en cinco secciones principales. En la primera se presenta la arquitectura general del sistema, describiendo los componentes que lo integran y su interacción. La segunda sección aborda el subsistema IoT, detallando el funcionamiento de los sensores, la transmisión de datos y el control automatizado del riego. La tercera describe la plataforma web, explicando su estructura de backend y frontend, así como las herramientas empleadas para la gestión de información y la comunicación con el hardware. En la cuarta se expone el

subsistema predictivo, donde se describe el modelo de aprendizaje automático utilizado para anticipar los niveles de humedad del suelo. Finalmente, en la quinta sección se presentan los procesos de integración y validación funcional del sistema, donde se analiza el desempeño general del prototipo y su capacidad para operar de manera autónoma y eficiente.

El enfoque adoptado permitió no solo demostrar el funcionamiento integral del sistema, sino también sentar las bases para su futura implementación en escenarios agrícolas reales. En conjunto, este capítulo representa la culminación del proceso metodológico descrito previamente, traduciendo los requerimientos identificados en un sistema tangible, funcional y validado experimentalmente.

4.1. Desarrollo del Sistema SIRCA-IoT

4.1.1. Arquitectura General del Sistema

El desarrollo del SIRCA-IoT se estructuró bajo una arquitectura distribuida y modular, diseñada para integrar de forma coherente los componentes de hardware, software y modelado predictivo. La arquitectura propuesta permitió el monitoreo continuo de variables ambientales, la automatización del riego y la generación de predicciones basadas en datos históricos, todo dentro de un entorno controlado que reproduce las condiciones de un cultivo protegido.

La solución se organizó en tres subsistemas principales: el subsistema IoT, encargado de la adquisición y transmisión de datos sensoriales; el subsistema web, responsable del procesamiento, almacenamiento, visualización y control del sistema; y el subsistema predictivo, dedicado al análisis de datos y la estimación de la humedad del suelo mediante aprendizaje automático. Estos subsistemas se comunicaron a través del protocolo MQTT (Message Queuing Telemetry Transport), que garantizó la transmisión eficiente y confiable de los datos entre los distintos nodos y servicios del sistema.

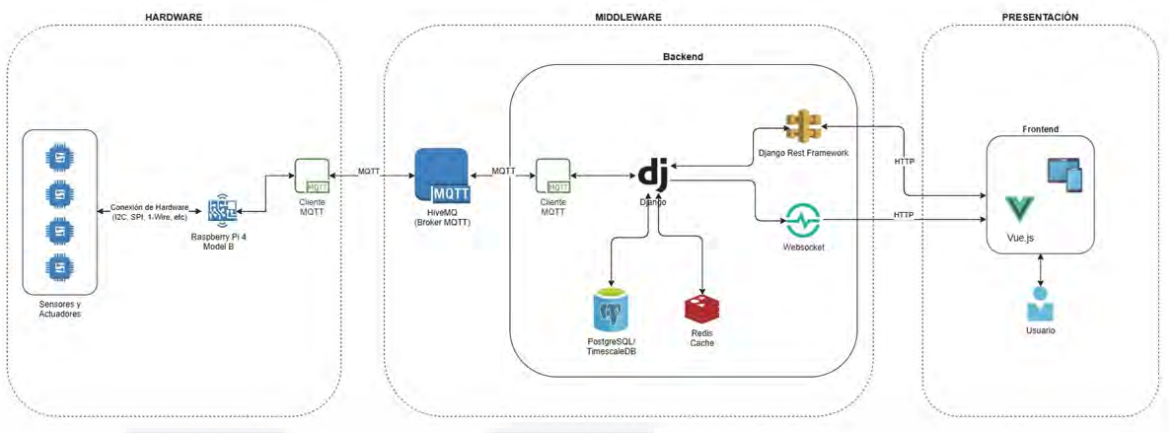


Diagrama 1. Arquitectura General del Sistema.

4.1.1.1. Estructura de la Arquitectura Distribuida

La arquitectura se basó en el principio de desacoplamiento funcional, permitiendo que cada subsistema operara de manera independiente pero sincronizada. La Raspberry Pi 4 Model B se utilizó como nodo central de adquisición y control del SIRCA-IoT, recibiendo los datos de los sensores conectados a sus interfaces GPIO, I2C y 1-Wire. Estos datos fueron procesados localmente y enviados al bróker MQTT HiveMQ, que actuó como intermediario entre el hardware y la plataforma web.

En el lado del servidor, el backend desarrollado en Django se encargó de recibir y registrar los datos en una base de datos TimescaleDB, optimizada para el manejo de series temporales. A partir de esta información, el sistema permitió tanto el monitoreo en tiempo real como el almacenamiento histórico para su posterior análisis. La comunicación con los usuarios se gestionó mediante una interfaz web desarrollada en Vue.js, la cual permitió la visualización gráfica de los datos, el control manual del riego y el seguimiento de las predicciones generadas por el modelo de machine learning.

El modelo predictivo, implementado y entrenado en Python, se integró al backend mediante un servicio modular que procesaba los datos registrados, estimaba la humedad futura del suelo y enviaba las recomendaciones al sistema de control. De esta manera, el flujo de información siguió un esquema circular que unió los

procesos de sensado, transmisión, predicción y acción. A continuación, se describen los componentes de cada uno de estos subsistemas:

- **Backend (Django + DRF + Channels):**

- **Django REST Framework (DRF):** Facilita la creación de endpoints para la interacción con el frontend, permitiendo la consulta de datos históricos de los sensores y la configuración de sus umbrales [31].
- **Django Channels:** Utilizado para gestionar la comunicación en tiempo real mediante WebSockets, lo que permite al frontend recibir actualizaciones instantáneas sobre los valores de los sensores sin necesidad de recargar la página [32].
- **Redis:** Actúa como cache para optimizar la velocidad de comunicación y como broker para las tareas asíncronas gestionadas por Celery [33].
- **Celery:** Se utiliza para gestionar tareas asíncronas, como la recolección periódica de datos y la integración con dispositivos de control, según las necesidades del sistema [34].

- **Frontend (Vue 3 + Vuetify 3):**

- **Vue 3:** Gestiona la reactividad de la interfaz, actualizando en tiempo real la visualización de los sensores y cualquier otra información relevante [35].
- **Vuetify 3:** Proporciona los componentes de interfaz de usuario necesarios para crear una experiencia visual atractiva y fácil de usar [36].
- **Vue3-ApexCharts:** Permite la visualización gráfica de los datos históricos de los sensores mediante gráficos interactivos [37].
- **Pinia:** Facilita el manejo del estado de la aplicación, permitiendo el almacenamiento de configuraciones y datos en el frontend [38].
- **Axios:** Se usa para interactuar con las APIs del backend, consultando datos históricos y enviando solicitudes para cambiar configuraciones o realizar acciones manuales [39].

➢ **Almacenamiento de Datos (TimescaleDB + PostgreSQL)**

- **TimescaleDB:** Permite almacenar y consultar datos en tiempo real y a lo largo del tiempo, optimizando las operaciones sobre grandes volúmenes de datos de sensores [40].
- **PostgreSQL:** Base de datos relacional utilizada para la gestión de los datos persistentes, como la configuración de los sensores, registros de usuarios, etc [41].

4.1.1.2. Comunicación y Flujo de Datos

El sistema se organiza de manera que cada componente se comunica de forma eficiente, permitiendo la actualización en tiempo real de los datos y la interacción con el usuario.

1. **Sensores → Backend:** Los datos de los sensores se envían al backend mediante **MQTT** o protocolos similares, para su procesamiento y análisis.
2. **Backend → Frontend:** A través de **WebSockets**, el backend envía actualizaciones en tiempo real a la interfaz de usuario, permitiendo al usuario ver los cambios en los valores de los sensores sin necesidad de recargar la página.
3. **Frontend → Backend (REST API):** El usuario puede interactuar con el sistema a través de la API RESTful, permitiendo la consulta de datos históricos y la modificación de configuraciones de los sensores.
4. **Backend → TimescaleDB:** Los datos de los sensores se almacenan en **TimescaleDB** para su posterior consulta.

Este diseño garantizó una comunicación bidireccional, en la que tanto los datos de sensores como las órdenes de control fluyeron de manera constante y sincrónica, lo que permitió mantener la operación autónoma del sistema con mínima intervención humana.

4.1.1.3. Principios de Diseño

Durante el desarrollo del sistema se adoptaron principios de ingeniería de software y hardware orientados a garantizar su fiabilidad, escalabilidad y sostenibilidad. Entre los más relevantes se destacaron los siguientes:

- Modularidad: cada componente del sistema fue diseñado como una unidad funcional independiente (sensores, backend, frontend, modelo predictivo), facilitando el mantenimiento, la depuración y la futura ampliación del prototipo.
- Escalabilidad: la arquitectura permitió la incorporación de nuevos sensores, nodos IoT o zonas de riego sin modificar el núcleo del sistema.
- Eficiencia energética: el hardware fue configurado para minimizar el consumo eléctrico, lo cual resulta esencial en aplicaciones agrícolas de bajo mantenimiento.
- Comunicación ligera: el uso del protocolo MQTT redujo la sobrecarga de datos y optimizó la transmisión en entornos de conectividad limitada.
- Seguridad y confiabilidad: se implementaron mecanismos de autenticación en el bróker MQTT y en la API web, garantizando la integridad de los datos y la protección frente a accesos no autorizados.

La modularidad del sistema facilita la incorporación de nuevas funcionalidades sin alterar el núcleo del sistema. Además, la capacidad de integrar dispositivos adicionales como válvulas de riego, sensores adicionales o sistemas de control más avanzados asegura que el sistema pueda adaptarse a distintas necesidades agrícolas [42].

Estos principios sustentaron el diseño general del sistema y guiaron su implementación en las fases posteriores, asegurando que el prototipo alcanzara un equilibrio entre complejidad técnica, eficiencia operativa y viabilidad práctica.

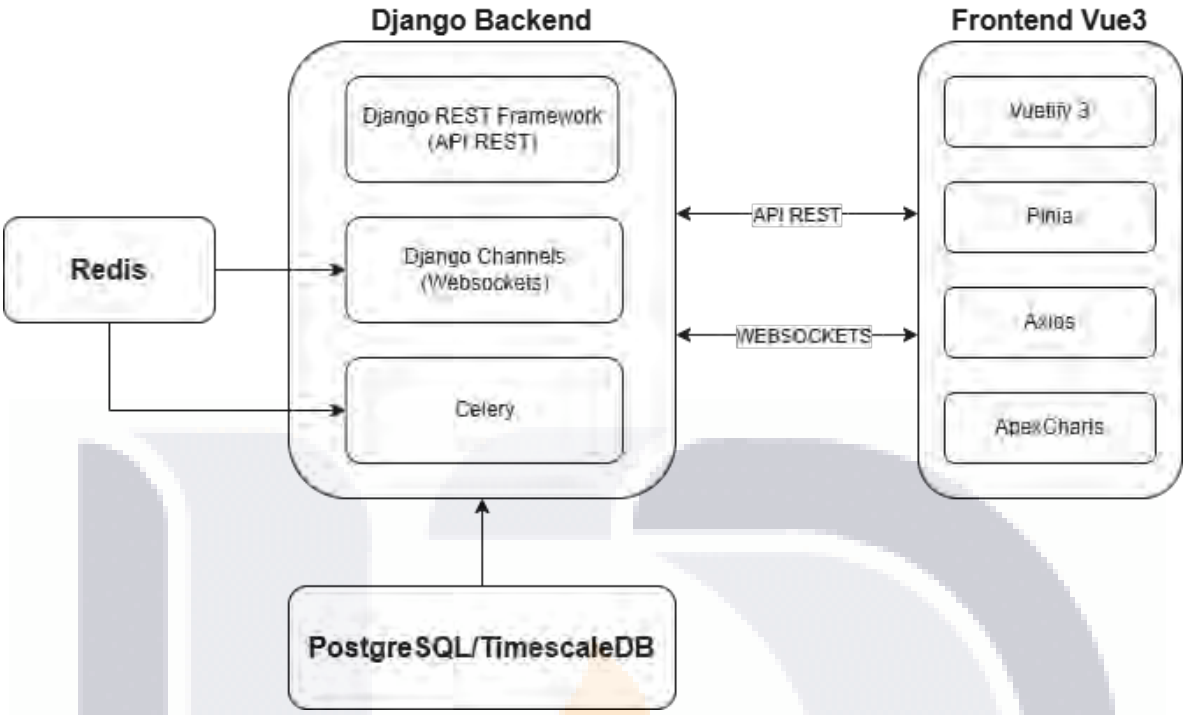


Diagrama 2. Arquitectura del Sistema (Backend, Frontend y Almacenamiento de Datos).

4.1.2. Subsistema IoT: Adquisición y Transmisión de Datos

El subsistema IoT constituyó la capa física del sistema de riego inteligente, encargada de la medición y transmisión de las variables ambientales que sirvieron como base para la toma de decisiones. Este componente se diseñó para recolectar información de forma continua y confiable, garantizando la comunicación con el servidor central mediante el protocolo MQTT. La integración de sensores con la Raspberry Pi permitió simular un entorno agrícola controlado, en el cual se validaron los procesos de adquisición, procesamiento y envío de datos hacia la plataforma web.

4.1.2.1. Controlador Central

El controlador central es el núcleo del sistema de riego inteligente basado en IoT. Su función principal es la adquisición de datos desde los sensores, el procesamiento

de la información, el control del actuador (bomba de agua) y la transmisión de datos a través del protocolo MQTT al broker de HiveMQ.

En este sistema, se ha seleccionado la Raspberry Pi 4 Model B como unidad central de procesamiento y comunicación debido a su capacidad de cómputo, conectividad y compatibilidad con múltiples sensores y dispositivos IoT [43].

Características Técnicas de la Raspberry Pi 4 Model B:

Procesador: Broadcom BCM2711, Quad-Core Cortex-A72 a 1.5 GHz

Memoria RAM: Variantes de 2GB, 4GB u 8GB LPDDR4

Almacenamiento: Tarjeta microSD (recomendada de 16GB o más)

Conectividad:

- Wi-Fi 802.11 b/g/n/ac
- Bluetooth 5.0
- Ethernet Gigabit
- 4 puertos USB (2x USB 3.0, 2x USB 2.0)
- GPIO de 40 pines para la conexión de sensores y actuadores [43]

Sistemas operativos compatibles: Raspberry Pi OS (recomendado), Ubuntu, entre otros [43].

La Raspberry Pi ejecutó un servicio en Python que gestionó las siguientes tareas principales:

- Lectura de datos desde los sensores conectados mediante los puertos GPIO, I2C y 1-Wire.
- Preprocesamiento local de las mediciones (filtrado, normalización y validación).

- Publicación de datos hacia el bróker MQTT, en tópicos definidos para cada variable.
- Recepción de comandos de control, enviados desde el backend para el encendido o apagado remoto de la bomba de agua.

Esta configuración permitió una comunicación bidireccional entre el hardware y el sistema web, reproduciendo el comportamiento esperado en un escenario agrícola real. La elección de la Raspberry Pi 4 Model B permite un alto grado de flexibilidad y escalabilidad en el sistema, garantizando una gestión eficiente del riego mediante el monitoreo en tiempo real y la comunicación con el broker MQTT de HiveMQ [44].

4.1.2.2. Sensores y Actuadores Implementados

4.1.2.2.1. Sensores

El sistema de riego inteligente basado en IoT incorpora una serie de sensores diseñados para recopilar información sobre el entorno y el suelo, permitiendo una gestión eficiente del agua. A continuación, se describen en detalle los sensores empleados, su funcionamiento y su integración con la Raspberry Pi 4 Model B.

DHT22 (Humedad y temperatura ambiental):

El sensor DHT22 es un dispositivo digital que mide la temperatura y la humedad relativa del aire. Es ampliamente utilizado en aplicaciones de monitoreo ambiental debido a su precisión y bajo consumo energético [45].

- Rango de temperatura: -40°C a 80°C
- Precisión de temperatura: ±0.5°C
- Rango de humedad relativa: 0% - 100%
- Precisión de humedad: ±2-5%
- Interfaz de comunicación: Digital, protocolo de un solo cable.

DS18B20 (Temperatura del suelo):

Este sensor digital es ideal para medir la temperatura del suelo debido a su encapsulado resistente al agua. Utiliza el protocolo 1-Wire, permitiendo la conexión de múltiples sensores en el mismo bus de datos [45].

- Rango de temperatura: -55°C a 125°C
- Precisión: ±0.5°C en el rango de -10°C a 85°C
- Interfaz de comunicación: 1-Wire
- Voltaje de operación: 3.0V - 5.5V

BH1750 (Intensidad de luz ambiental):

El sensor BH1750 mide la cantidad de luz en lux y proporciona datos en formato digital mediante la interfaz I²C. Es crucial para determinar la influencia de la luz en la evaporación del agua en el suelo [45].

- Rango de medición: 1 - 65535 lux
- Interfaz de comunicación: I²C
- Voltaje de operación: 2.4V - 3.6V
- Precisión: ±20%

SCD41 (Sensor de CO_2):

Este sensor mide la concentración de dióxido de carbono en el aire, lo cual es útil para analizar el impacto de la ventilación y la fotosíntesis en el invernadero [45].

- Rango de medición: 0 - 40,000 ppm
- Interfaz de comunicación: I²C
- Voltaje de operación: 2.4V - 5.5V

- Precisión: ±(50 ppm + 5% del valor medido)

LM393 (Sensor de humedad del suelo con ADC ADS1115):

Este sensor, combinado con el convertidor ADC ADS1115, permite medir la humedad del suelo de manera precisa y enviarla a la Raspberry Pi, ya que la Raspberry Pi no cuenta con entradas analógicas nativas [45].

- Interfaz del sensor: Analógica
- Conversión ADC: ADS1115 (16 bits)
- Voltaje de operación: 3.3V - 5V
- Precisión de medición: Alta, debido al uso del convertidor ADC

Cada sensor fue probado de manera individual para verificar su correcta lectura, y posteriormente integrado al sistema general. Las calibraciones y conexiones eléctricas se documentaron en el Anexo A, donde se describen los esquemas de cableado y los parámetros de configuración empleados durante las pruebas.

4.1.2.2.2. Actuadores

El sistema de riego inteligente basado en IoT cuenta con un único actuador: una bomba de agua controlada mediante un relé. Este componente es el encargado de regular el suministro de agua a los cultivos en función de los valores de humedad del suelo obtenidos a través del sensor LM393, garantizando un uso eficiente del recurso hídrico.

Características de la bomba de agua:

- Tipo: Bomba de agua sumergible.
- Voltaje de operación: 3V - 6V.
- Caudal de agua: 80-120 L/h (2 litros por minuto)

- Modo de activación: Controlada por un relé mediante la Raspberry Pi

Módulo Relé

Para controlar la activación de la bomba de agua con la Raspberry Pi, se emplea un módulo relé de estado sólido o mecánico que actúa como un interruptor electrónico, permitiendo la conexión o desconexión de la alimentación de la bomba.

- Voltaje de control: 3.3V - 5V
- Voltaje de carga: Hasta 250V AC o 30V DC
- Corriente soportada: 10A
- Tipo de relé: SPDT (Single Pole Double Throw)

Este actuador es un elemento clave en la automatización del riego, ya que permite gestionar el flujo de agua de manera inteligente en respuesta a las condiciones del suelo, optimizando así el consumo de agua y mejorando la salud del cultivo [45]. Las calibraciones y conexiones eléctricas se documentaron en el Anexo A, donde se describen los esquemas de cableado y los parámetros de configuración empleados durante las pruebas.

4.1.2.3. Conectividad y Comunicación

La conectividad y comunicación en el sistema de riego inteligente basado en IoT son fundamentales para garantizar la transmisión eficiente y en tiempo real de los datos obtenidos por los sensores y las órdenes de activación del actuador. Para ello, se utilizó el protocolo Message Queuing Telemetry Transport (MQTT) y el servicio de broker MQTT de HiveMQ, que permite la comunicación entre la Raspberry Pi y los demás dispositivos del sistema [46].

4.1.2.3.1. Protocolo MQTT

MQTT es un protocolo de mensajería liviano ideal para sistemas IoT debido a su baja latencia, consumo mínimo de ancho de banda y fiabilidad en la transmisión de

datos. Funciona bajo un modelo publicador-suscriptor, en el cual los dispositivos pueden publicar y recibir mensajes en distintos "tópicos" dentro del broker.

- **Modelo de Comunicación:** Publicador-Suscriptor
- **Protocolo de Transporte:** TCP/IP
- **Seguridad:** Compatible con autenticación TLS/SSL
- **Eficiencia:** Bajo consumo de ancho de banda, ideal para IoT

En este sistema, la Raspberry Pi actúa como publicador y suscriptor, enviando datos de los sensores y recibiendo comandos para la activación de la bomba de agua [47].

4.1.2.3.2. Broker MQTT de HiveMQ

HiveMQ es un broker MQTT basado en la nube que facilita la comunicación segura y confiable entre dispositivos IoT. Se ha elegido este servicio debido a su estabilidad, facilidad de integración y soporte para múltiples clientes simultáneos.

Características principales:

- Soporte para conexiones simultáneas de múltiples dispositivos
- Baja latencia en la transmisión de mensajes
- Seguridad con TLS/SSL y autenticación de clientes
- Compatible con QoS (Calidad de Servicio) para priorización de mensajes [48].

4.1.2.3.3. Tópicos MQTT Utilizados en el Sistema

En este sistema se han definido los siguientes tópicos MQTT para la transmisión de datos y el control del actuador:

- **Tópico de Sensores: iot/riego/sensores**

Publica los valores de temperatura, humedad ambiental, humedad del suelo, CO_2 e intensidad lumínica.

- **Tópico de Control de la Bomba: iot/riego/bomba**

  Recibe órdenes de activación o desactivación de la bomba de agua.

4.1.2.3.4. Flujo de Datos y Procesamiento

El sistema de riego inteligente basado en IoT sigue un flujo de datos bien definido para garantizar la correcta adquisición, procesamiento y transmisión de información. A continuación, se detalla el flujo de datos en el sistema:

1. Adquisición de Datos:
   - Los sensores conectados a la Raspberry Pi recopilan información en tiempo real sobre la temperatura y humedad del ambiente (DHT22), la temperatura del suelo (DS18B20), la intensidad de luz (BH1750), los niveles de CO_2 (SCD41) y la humedad del suelo (LM393 con ADS1115).
   - Cada sensor tiene un script independiente en Python que ejecuta su lectura y retorna los valores obtenidos.
2. Procesamiento Local:
   - La Raspberry Pi recibe los datos desde los sensores y los convierte en un formato estructurado.
3. Transmisión de Datos al Broker MQTT:
   - Todos los datos recopilados se publican en el broker HiveMQ a través del protocolo MQTT.

- La Raspberry Pi envía los datos a tópicos específicos en el broker:
    - iot/sensores: Contiene los valores de los sensores.
- El cliente MQTT también se suscribe al tópico iot/control para recibir comandos de activación y desactivación de la bomba de agua.

4. Interacción con el Usuario:
    - Los datos enviados al broker MQTT pueden ser accedidos por clientes remotos que deseen monitorear las condiciones del cultivo en tiempo real.
    - Se pueden enviar comandos desde una interfaz de usuario o aplicación para activar o desactivar manualmente la bomba de agua a través del broker MQTT.
5. Respuesta del Sistema:
    - Cuando se recibe un comando en el tópico iot/control, la Raspberry Pi lo interpreta y activa o desactiva la bomba de agua en consecuencia.
    - Se actualiza el estado del sistema en el broker MQTT, asegurando que cualquier cliente suscrito reciba la información más reciente.

Este flujo de datos garantiza la automatización eficiente del riego, optimizando el consumo de agua y asegurando condiciones óptimas para el crecimiento del cultivo.

El detalle completo de configuración y código fuente se presenta en el Anexo A.

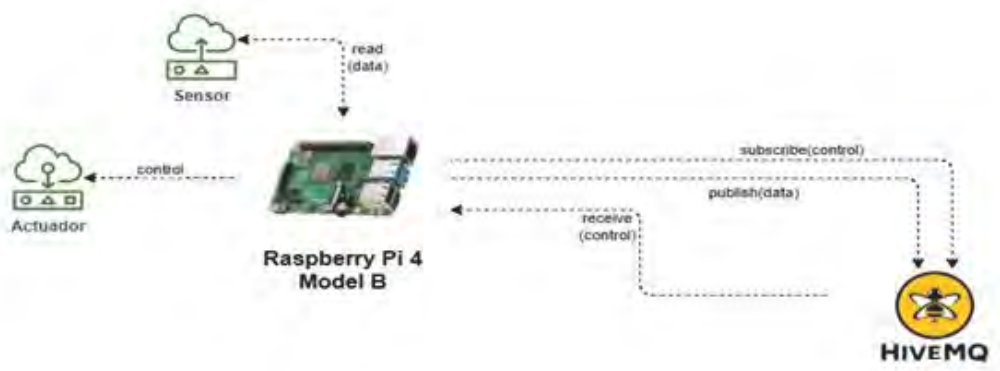


Diagrama 3. Diagrama de flujo de datos y procesamiento en el sistema IoT.

4.1.3. Subsistema Web: Backend y Frontend

El subsistema web constituyó la capa lógica y de presentación del sistema de riego inteligente, encargada de recibir, procesar, almacenar y mostrar la información proveniente de los dispositivos IoT, así como de administrar la comunicación con el modelo predictivo y los módulos de control. Este subsistema se diseñó bajo una arquitectura cliente-servidor que integró tecnologías modernas de desarrollo web, garantizando un entorno robusto, escalable y de fácil uso para el operador.

Este sistema está compuesto por dos módulos principales: un backend desarrollado con Django, encargado de recibir, procesar y almacenar datos desde los sensores; y un frontend construido en Vue 3, que permite visualizar la información en tiempo real, explorar métricas históricas y configurar alertas o parámetros del sistema. La arquitectura integra tecnologías como MQTT para la comunicación IoT, WebSockets para actualización en tiempo real, y TimescaleDB para el almacenamiento de series temporales, gestionando procesos asíncronos mediante Celery.

En esta sección se describen los componentes técnicos del sistema web, el flujo de datos desde su origen hasta la visualización, y los mecanismos implementados para garantizar su escalabilidad, modularidad y seguridad.

4.1.3.1. Backend del Sistema

El backend fue desarrollado utilizando el framework Django junto con Django REST Framework (DRF), debido a su estabilidad, seguridad y facilidad para estructurar aplicaciones modulares. Este componente funcionó como el núcleo del sistema, encargándose de la comunicación entre la Raspberry Pi, la base de datos, el modelo predictivo y la interfaz web.

El backend gestionó las siguientes funciones principales:

- Recepción y almacenamiento de datos IoT:
  Los datos enviados desde la Raspberry Pi a través del bróker MQTT fueron recibidos mediante un cliente integrado en el backend, que procesó los mensajes y los almacenó en una base de datos TimescaleDB. Este sistema, diseñado para manejar series temporales, permitió conservar registros históricos y realizar consultas eficientes sobre grandes volúmenes de datos ambientales.
- Exposición de servicios mediante API RESTful:
  Se desarrollaron endpoints que facilitaron la comunicación entre los distintos módulos del sistema. A través de estas rutas, el frontend accedió a los datos de sensores, al historial de humedad del suelo y a las predicciones generadas por el modelo de machine learning. Las rutas principales incluyeron funciones para la consulta, control manual del riego, gestión de dispositivos y monitoreo de eventos.
- Integración con el modelo predictivo:
  El backend alojó un servicio encargado de ejecutar el modelo de predicción de humedad del suelo. Este proceso analizaba las últimas lecturas almacenadas y generaba una estimación del nivel de humedad futuro. Los resultados eran devueltos al sistema central y mostrados en la interfaz de usuario como sugerencias de riego.
- Control remoto del riego:
  Cuando el sistema recibía una orden de activación o desactivación del riego (ya sea de manera automática o manual), generaba un mensaje MQTT hacia

el tópico correspondiente, que era recibido por la Raspberry Pi. Este proceso cerró el ciclo de comunicación entre software y hardware.

Para optimizar el rendimiento y la ejecución de tareas periódicas, se integró un sistema de procesamiento asíncrono con Celery y Redis, utilizado para tareas como el envío recurrente de datos de diagnóstico o el cálculo de predicciones a intervalos definidos. La configuración detallada de estos servicios, junto con los scripts de integración MQTT, se documenta en el Anexo B.

4.1.3.1.1. Stack Tecnológico

El backend del sistema se desarrolla utilizando una combinación de tecnologías robustas y escalables. A continuación, se detallan las principales tecnologías utilizadas en el backend para garantizar un alto rendimiento, facilidad de mantenimiento y capacidad de ampliación.

- **Django**

  Django es un framework web de alto nivel que facilita el desarrollo de aplicaciones web complejas de manera rápida y con un enfoque en la reutilización de código. Django es conocido por su enfoque en la "convención sobre configuración", lo que significa que proporciona una estructura clara y bien definida para que los desarrolladores puedan concentrarse en la lógica de negocio sin tener que preocuparse por los detalles técnicos de bajo nivel.

  Ventajas de usar Django:

  - Escalabilidad: Django es altamente escalable, lo que permite que el sistema crezca con facilidad a medida que se añaden nuevos sensores, dispositivos o funcionalidades.
  - Seguridad: Django incluye múltiples medidas de seguridad integradas, como protección contra CSRF (Cross-Site Request Forgery), XSS (Cross-Site Scripting) y SQL Injection.
  - ORM (Object-Relational Mapping): Django utiliza un ORM para interactuar con bases de datos, lo que permite definir modelos de datos como clases Python y gestionarlas de manera sencilla sin tener que escribir consultas SQL directamente [49].

- **Django REST Framework (DRF)**

  Django REST Framework (DRF) es una extensión de Django diseñada para crear APIs RESTful de manera rápida y sencilla. DRF permite exponer los recursos del backend (como las lecturas de los sensores, configuraciones de los dispositivos y notificaciones) a través de endpoints que pueden ser consumidos por el frontend.

  Ventajas de usar DRF:

  - Facilidad para crear APIs RESTful: Con DRF, la creación de una API es rápida y sencilla gracias a sus herramientas como serializers y viewsets.
  - Autenticación y permisos: DRF proporciona un sistema de autenticación y autorización flexible, permitiendo aplicar permisos específicos a cada recurso (por ejemplo, solo ciertos usuarios pueden modificar la configuración de los sensores).
  - Documentación automática: DRF incluye herramientas para generar documentación de la API de manera automática, facilitando la interacción entre el backend y el frontend [31].

- **Django Channels**

  Django Channels extiende la capacidad de Django para manejar WebSockets, lo que permite la comunicación en tiempo real entre el backend y el frontend. En el caso de este sistema, se utiliza Django Channels para permitir que los datos de los sensores y las notificaciones de control, como el estado de la bomba de agua, se transmitan en tiempo real al frontend.

  Ventajas de usar Django Channels:

  - Comunicación en tiempo real: Con WebSockets, el backend puede enviar datos al frontend sin necesidad de que el usuario recargue la página, lo que mejora la experiencia de usuario al ver la información en tiempo real.

- Soporte para múltiples protocolos: Django Channels no solo soporta WebSockets, sino también protocolos como HTTP2, lo que amplía las capacidades del sistema para otros tipos de comunicación en el futuro [32].

➢ **Redis**

Redis es un sistema de almacenamiento en memoria que se utiliza como cache y como broker de tareas. En este sistema, Redis cumple dos roles fundamentales:

- Cache: Redis se utiliza para almacenar temporalmente los resultados de consultas frecuentes o datos que no cambian con frecuencia, lo que mejora la velocidad de acceso a esos datos y reduce la carga en la base de datos.
- Broker para Celery: Redis se utiliza como el broker para Celery, facilitando la ejecución de tareas asíncronas. Por ejemplo, la recolección periódica de datos de los sensores o la ejecución de procesos de control, como el monitoreo y activación de la bomba de agua, se gestionan a través de Celery utilizando Redis.

Ventajas de usar Redis:

- Alta velocidad: Redis es extremadamente rápido debido a que almacena datos en memoria, lo que lo hace ideal para operaciones de cache.
- Escalabilidad: Redis es fácil de escalar horizontalmente, lo que permite distribuir el procesamiento de tareas en múltiples nodos para mejorar el rendimiento y la capacidad de gestión de la carga [33].

➢ **Celery**

Celery es una librería para la gestión de tareas asíncronas. Se utiliza en este sistema para ejecutar tareas que no necesitan ser procesadas

inmediatamente, como la recolección periódica de datos de los sensores y el control de la bomba de agua. Celery gestiona estas tareas en segundo plano, permitiendo que el sistema siga funcionando sin bloquearse mientras se realizan estas operaciones.

Ventajas de usar Celery:

- Tareas asíncronas y periódicas: Celery permite que tareas de larga duración o repetitivas, como la recolección de datos de sensores o el monitoreo de la bomba de agua, se realicen de manera asíncrona sin afectar el rendimiento del sistema.
- Escalabilidad: Celery se puede ejecutar en varios trabajadores (workers), lo que permite distribuir las tareas y mejorar la capacidad de procesamiento.
- Soporte para diferentes tipos de colas: Celery puede integrarse con diferentes sistemas de colas de mensajes, y en este caso, se utiliza con Redis como bróker [34], [50].

- **TimescaleDB**

TimescaleDB es una extensión de PostgreSQL diseñada específicamente para almacenar y gestionar series temporales. Dado que los datos de los sensores son, por naturaleza, temporales (cambian continuamente a lo largo del tiempo), TimescaleDB es ideal para almacenar y consultar este tipo de datos de manera eficiente.

Ventajas de usar TimescaleDB:

- Optimización para datos temporales: TimescaleDB permite realizar consultas complejas sobre grandes volúmenes de datos temporales de manera eficiente.
- Integración con PostgreSQL: TimescaleDB se integra de forma nativa con PostgreSQL, lo que permite aprovechar todas las funcionalidades de una base de datos relacional con optimizaciones para series temporales [40], [41].

El detalle de la implementación web se presenta en el Anexo B.

4.1.3.1.2. Recepción de Datos desde HiveMQ

El sistema se comunica con los sensores a través de un bróker MQTT, que es responsable de recibir, gestionar y distribuir los mensajes de los sensores. El bróker HiveMQ es utilizado en este sistema como intermediario entre los sensores y el backend, permitiendo que los datos generados por los sensores se transmitan de manera eficiente hacia el backend para su procesamiento y almacenamiento.

Protocolo MQTT y HiveMQ

MQTT es un protocolo ligero de mensajería que permite la comunicación entre dispositivos de bajo consumo, como los sensores en un cultivo, y sistemas más complejos como el backend. MQTT se basa en un modelo de publicación y suscripción, lo que significa que los sensores (publicadores) envían datos a través de un canal (llamado "tema") al que el backend (suscriptor) se suscribe para recibir los mensajes.

El bróker HiveMQ actúa como el intermediario en este proceso:

- Los sensores publican mensajes con datos a un tema específico.
- El backend se suscribe a esos temas y recibe los mensajes de los sensores en tiempo real [47], [48].

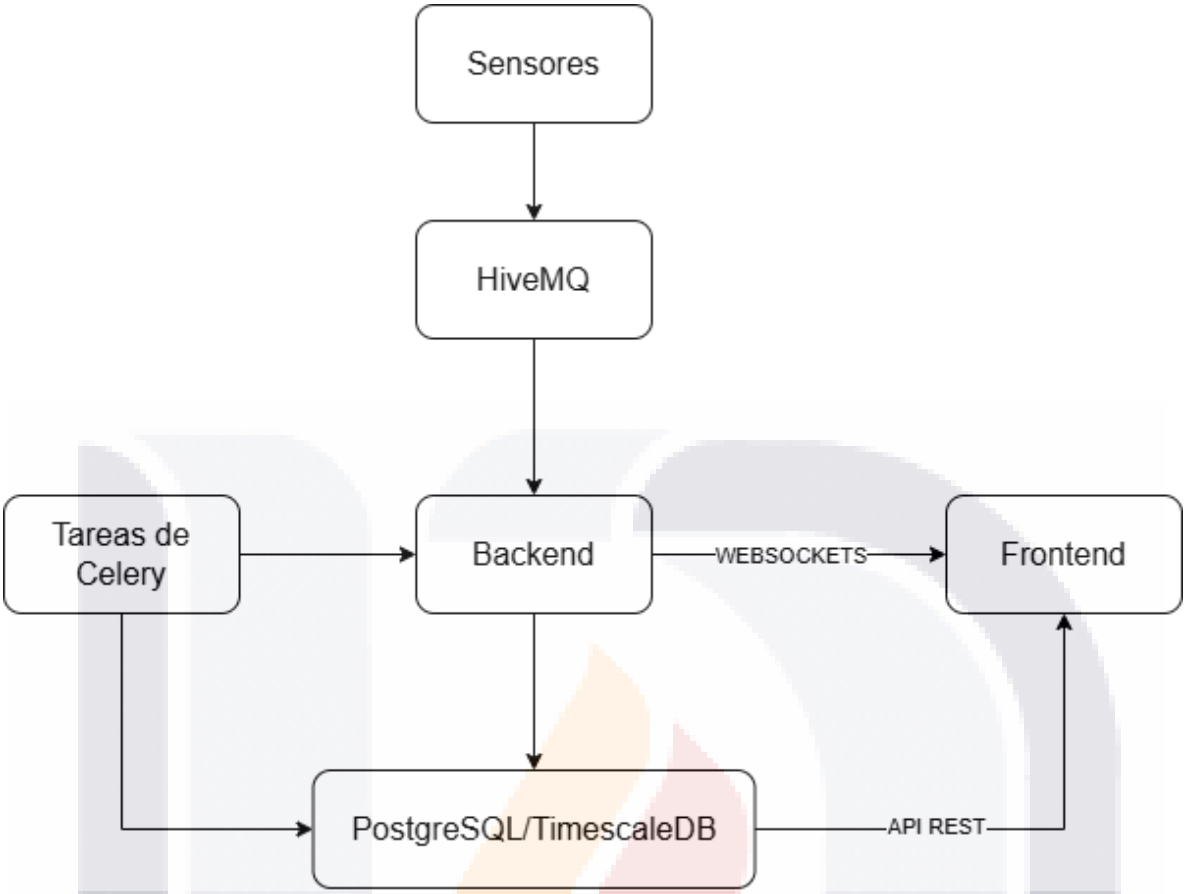


Diagrama 4. Diagrama de Flujo de Datos en el Backend.

4.1.3.1.3. Envío de Datos en Tiempo Real (WebSocket)

En un sistema de monitoreo en tiempo real, es fundamental que el frontend se mantenga actualizado con la información más reciente, sin necesidad de recargar la página. Para lograr esto, se utiliza la tecnología WebSocket, que permite la comunicación bidireccional en tiempo real entre el servidor (backend) y el cliente (frontend). En este caso, se emplea Django Channels para gestionar las conexiones WebSocket y enviar las actualizaciones de los sensores al frontend de manera instantánea.

Uso de Django Channels para WebSockets:

Django Channels extiende la funcionalidad de Django para soportar protocolos asíncronos, como WebSockets, lo que permite gestionar conexiones de larga

duración entre el servidor y el cliente. A diferencia de las solicitudes HTTP tradicionales, que son transacciones de ida y vuelta, los WebSockets permiten que los datos fluyan continuamente entre el servidor y el cliente sin necesidad de nuevas solicitudes.

Flujo de Comunicación con WebSocket

1. Conexión del cliente WebSocket al servidor: El cliente (en este caso, el frontend desarrollado en Vue 3) establece una conexión WebSocket con el backend.
2. Backend recibe los datos de los sensores: El backend procesa los datos de los sensores, los cuales son enviados en tiempo real a través del WebSocket.
3. Actualización en tiempo real del frontend: El frontend, a través de la conexión WebSocket, recibe los datos y actualiza la interfaz de usuario sin necesidad de recargar la página [32].

4.1.3.1.4. Tareas Periódicas con Celery

El uso de Celery en el sistema es clave para manejar tareas que deben ejecutarse de forma periódica, sin bloquear la ejecución del sistema principal. Esto es especialmente útil en un sistema de monitoreo como el de esta tesis, donde se requiere la recolección periódica de datos de los sensores, el control de dispositivos (como la bomba de agua) o la ejecución de otros procesos que deben suceder en segundo plano, sin que el sistema deje de responder a las solicitudes del usuario.

Celery es un framework para tareas distribuidas en Python, diseñado para la ejecución de tareas en segundo plano. En este sistema, Celery se utiliza en conjunto con Celery Beat, que se encarga de programar las tareas periódicas, como la recolección de datos a intervalos regulares o la gestión de dispositivos [34], [50].

4.1.3.1.5. API REST para Configuración

Una de las funcionalidades clave del backend en este sistema es proporcionar una interfaz para que los usuarios gestionen las configuraciones de los sensores, como

los umbrales de activación para la bomba de agua, las métricas que se deben monitorear, y otros parámetros del sistema. Para ello, se utiliza una API REST expuesta mediante Django REST Framework (DRF), lo que facilita la interacción con el frontend y otros sistemas.

- **Creación de la API REST con Django REST Framework**

Django REST Framework (DRF) es una poderosa herramienta que permite crear rápidamente APIs RESTful en Django. DRF facilita la creación de serializadores para transformar los modelos de Django en representaciones JSON, y también proporciona vistas para manejar las solicitudes HTTP [31].

Configuración de Django REST Framework (DRF):

Primero, instalamos DRF en nuestro entorno de trabajo:

```
pip install djangorestframework
```

A continuación, agregamos DRF a las aplicaciones instaladas en el archivo `settings.py`:

```
# settings.py

INSTALLED_APPS = [
    # Otras aplicaciones
    'rest_framework',
]
```

Código 1. Configuración de Django REST Framework en settings.py.

- **Serializadores en DRF**

Los serializadores en DRF son responsables de transformar los objetos de Django (como los modelos) en datos JSON que pueden ser enviados a través de la API, y viceversa. Para la configuración de los sensores y otros parámetros del sistema, necesitamos crear un serializador para cada modelo relevante.

Ejemplo de Serializador para Configuración de Sensores:

En el siguiente ejemplo, creamos un serializador para la configuración de los sensores, como los umbrales de humedad y temperatura.

```python
# serializers.py
from rest_framework import serializers
from .models import SensorConfig

class SensorConfigSerializer(serializers.ModelSerializer):
    class Meta:
        model = SensorConfig
        fields = ['id', 'sensor_type', 'min_value', 'max_value']
```

Código 2: Ejemplo de Serializador para Configuración de Sensores.

- **`sensor_type`**: El tipo de sensor (temperatura, humedad, etc.).
- **`min_value` y `max_value`**: Los umbrales de valor para cada tipo de sensor (por ejemplo, el umbral mínimo y máximo de humedad) [31].

➢ **Vistas de la API**

Con el serializador definido, creamos las vistas que manejarán las solicitudes HTTP (GET, POST, PUT, DELETE) para obtener, crear, actualizar y eliminar las configuraciones de los sensores.

Ejemplo de Vista para Configuración de Sensores:

Usamos `ViewSets` en DRF para crear vistas fácilmente que gestionen las operaciones CRUD sobre los recursos de configuración.

```python
# views.py
from rest_framework import viewsets
from .models import SensorConfig
from .serializers import SensorConfigSerializer

class SensorConfigViewSet(viewsets.ModelViewSet):
    queryset = SensorConfig.objects.all()
    serializer_class = SensorConfigSerializer
```

Código 3: Ejemplo de Vista para Configuración de Sensores.

`SensorConfigViewSet`: Este `ViewSet` permite realizar operaciones CRUD sobre las configuraciones de los sensores. Con `ModelViewSet`, se generan automáticamente las vistas para las operaciones básicas (GET, POST, PUT, DELETE) [31].

- **Configuración de Rutas para la API REST**

Una vez que hemos creado los serializadores y las vistas, es necesario configurar las rutas de la API para que los usuarios y el frontend puedan acceder a ellas.

Configuración de Rutas en `urls.py`:

```python
# urls.py
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import SensorConfigViewSet

router = DefaultRouter()
router.register(r'sensor-config', SensorConfigViewSet)

urlpatterns = [
    path('api/', include(router.urls)),  # Rutas de la API
]
```

Código 4: Configuración de Rutas en urls.py.

En este caso, hemos creado una ruta `sensor-config` que permitirá a los usuarios consultar y modificar la configuración de los sensores. El `DefaultRouter` de DRF

genera automáticamente las rutas para las operaciones CRUD sobre los recursos [31].

- **Autenticación y Autorización en la API**

Es importante implementar un sistema de autenticación y autorización para asegurar que solo los usuarios con permisos adecuados puedan modificar la configuración de los sensores.

Autenticación con JWT:

Una de las opciones más comunes es el uso de JWT (JSON Web Tokens) para autenticar las solicitudes a la API. Para ello, podemos instalar una librería como **`djangorestframework-simplejwt`**.

```
pip install djangorestframework-simplejwt
```

En **`settings.py`**, configuramos JWT para autenticar las solicitudes:

```
# settings.py
INSTALLED_APPS = [
    # Otras aplicaciones
    'rest_framework_simplejwt',
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
}
```

Código 5: Configuración de `djangorestframework-simplejwt` en `settings.py`.

Con esta configuración, las solicitudes a la API deben incluir un token JWT válido en el encabezado para ser autorizadas [31].

4.1.3.2. Estructura de la Base de Datos

El sistema de monitoreo de sensores utiliza TimescaleDB para almacenar y gestionar los datos de los sensores y sus configuraciones. Dado que los datos son principalmente series temporales, como las mediciones periódicas de temperatura, humedad y otros parámetros ambientales, TimescaleDB se ha elegido como base de datos debido a su optimización para este tipo de datos [40].

4.1.3.2.1. Modelo de Datos en TimescaleDB

El modelo de datos que se utiliza para almacenar la información de los sensores está basado en varias tablas relacionadas entre sí. A continuación, se describe cada una de las tablas y su relación:

Tablas Principales:

1. **`sensors_sensor_type`**
   - **`id`**: UUID (Identificador único del tipo de sensor).
   - **`name`**: Nombre del tipo de sensor (por ejemplo, temperatura, humedad).
   - **`unit`**: Unidad de medición (por ejemplo, °C, %, etc.).
2. **`sensors_sensor`**
   - **`id`**: UUID (Identificador único del sensor).
   - **`name`**: Nombre del sensor (por ejemplo, "Sensor de humedad").
   - **`min_value`**: Valor mínimo permitido para el sensor.
   - **`max_value`**: Valor máximo permitido para el sensor.
   - **`location`**: Ubicación del sensor.
   - **`description`**: Descripción adicional sobre el sensor.
   - **`type_id`**: Relación con la tabla `sensors_sensor_type` (tipo de sensor).

3. **sensors_sensordata**
    - **time**: Timestamp (fecha y hora de la medición del sensor).
    - **value**: Valor de la medición del sensor.
    - **sensor_id**: Relación con la tabla `sensors_sensor` (sensor que realizó la medición).
4. **sensors_notification**
    - **id**: UUID (Identificador único de la notificación).
    - **status**: Estado de la notificación (por ejemplo, "activado", "desactivado").
    - **message**: Mensaje asociado con la notificación (por ejemplo, "El sensor de temperatura alcanzó el valor máximo").
    - **timestamp**: Timestamp (fecha y hora de la notificación).
    - **sensor_id**: Relación con la tabla `sensors_sensor` (sensor asociado a la notificación).

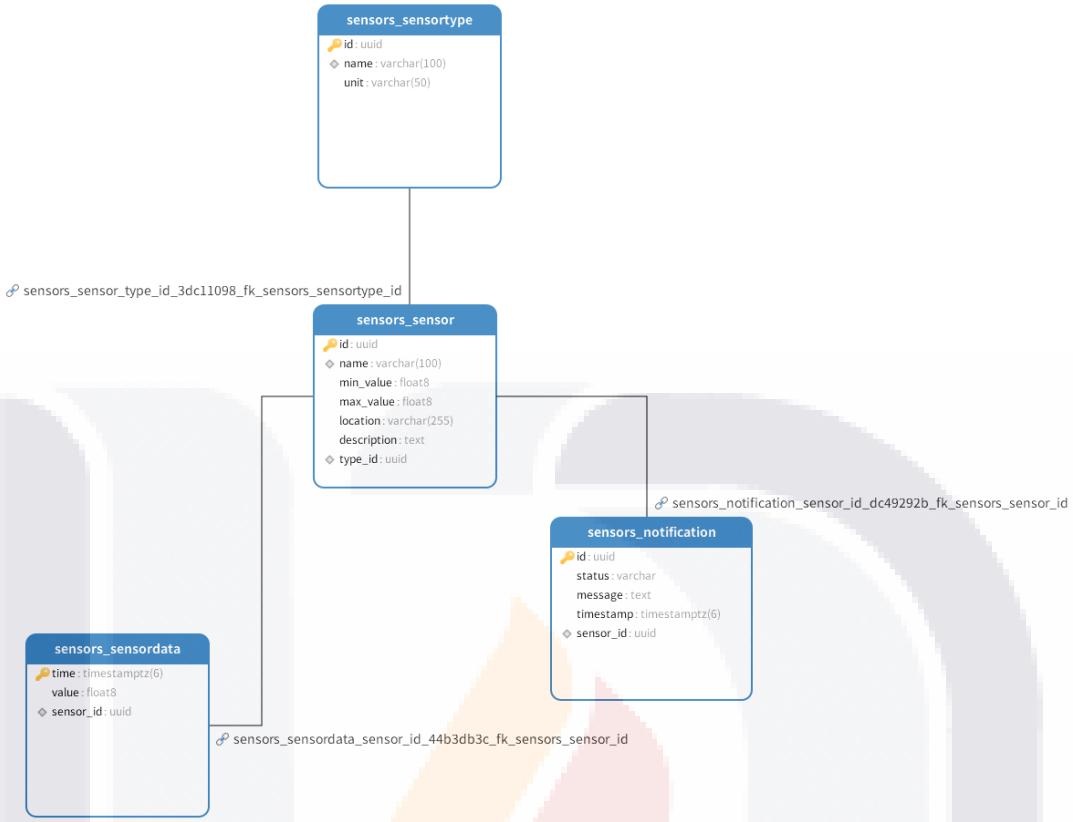


Imagen 1. Modelo de Datos (Diagrama ER Simplificado).

4.1.3.3. Frontend del Sistema

El frontend del sistema de monitoreo de sensores tiene como objetivo proporcionar una interfaz de usuario interactiva y eficiente, permitiendo la visualización en tiempo real de los datos de los sensores, la configuración de sus umbrales, y la gestión de notificaciones y métricas. El sistema está basado en una Single Page Application (SPA), lo que significa que toda la interacción con el usuario se realiza sin necesidad de recargar la página, lo que ofrece una experiencia más fluida y rápida [51].

Este apartado se centra en la tecnología y arquitectura utilizadas en el desarrollo del frontend, destacando las herramientas y los componentes principales que conforman la interfaz de usuario.

4.1.3.3.1. Stack Tecnológico

El frontend del sistema está desarrollado utilizando una combinación de tecnologías modernas que permiten crear una interfaz de usuario dinámica, interactiva y eficiente. Estas tecnologías incluyen Vue 3, Vuetify, Axios y Pinia, las cuales se integran perfectamente para ofrecer una experiencia fluida y escalable.

- **Vue 3**

  Vue.js es un framework progresivo de JavaScript utilizado para construir interfaces de usuario interactivas y dinámicas. Vue 3 es la versión más reciente, que introduce mejoras de rendimiento, composición y optimización en la reactividad del sistema. Vue 3 se utiliza en el frontend del sistema para crear una SPA, lo que significa que la aplicación se carga una vez y las interacciones se realizan sin recargar la página.

  Características Clave de Vue 3:

  - Reactividad: Vue 3 proporciona un sistema de reactividad eficiente que permite que la interfaz se actualice automáticamente cuando cambian los datos.
  - Composición API: Vue 3 introduce la Composition API, que permite organizar el código de manera más modular y reutilizable, facilitando el mantenimiento de la aplicación.
  - Componentes: Vue 3 se basa en componentes que permiten desarrollar de forma modular y mantener las distintas secciones de la aplicación como bloques independientes y reutilizables.

```vue
<template>
  <div>
    <h1>{{ title }}</h1>
    <p>{{ message }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      title: "Dashboard",
      message: "Bienvenido al sistema de monitoreo de sensores"
    };
  }
};
</script>

<style scoped>
h1 {
  color: #4CAF50;
}
</style>
```

Código 6: Ejemplo de un Componente en Vue 3.

Este es un ejemplo básico de un componente en Vue 3, que muestra un título y un mensaje. Los componentes en Vue permiten encapsular funcionalidad y diseño, lo que facilita el desarrollo y la gestión del código [35].

- **Vuetify**

  Vuetify es un framework de componentes basado en Material Design para Vue.js. Proporciona una colección de componentes predefinidos que facilitan la creación de interfaces de usuario atractivas y coherentes con los principios de diseño de Google.

  Características Clave de Vuetify:

  - Componentes de UI listos para usar: Vuetify incluye una amplia gama de componentes como botones, formularios, tablas, menús, etc., que siguen las pautas de Material Design.
  - Personalización fácil: Aunque Vuetify proporciona un conjunto predeterminado de componentes y estilos, también permite personalizar completamente la apariencia de la aplicación.

- Responsividad: Los componentes de Vuetify son completamente responsivos, lo que significa que se adaptan automáticamente a diferentes tamaños de pantalla y dispositivos.

```
<template>
  <v-container>
    <v-row>
      <v-col cols="12" sm="6">
        <v-card>
          <v-card-title>Configuración de Sensores</v-card-title>
          <v-card-text>
            Configura los umbrales de los sensores para activar la bomba de agua.
          </v-card-text>
        </v-card>
      </v-col>
    </v-row>
  </v-container>
</template>
```

Código 7: Ejemplo de Uso de Vuetify en un Componente.

En este ejemplo, usamos el componente v-container de Vuetify para organizar la disposición de la página. v-card se utiliza para crear una tarjeta con título y texto, que es común en las interfaces de usuario modernas [36].

- **Axios**

  Axios es una librería de JavaScript que se utiliza para realizar peticiones HTTP. En el frontend del sistema, Axios se utiliza para interactuar con las APIs REST del backend, permitiendo obtener datos como la configuración de los sensores, los datos históricos y las notificaciones, entre otros.

  Características Clave de Axios:

  - Basado en Promesas: Axios utiliza promesas para manejar las respuestas, lo que facilita el manejo de operaciones asíncronas.
  - Soporte para solicitudes HTTP: Permite realizar peticiones GET, POST, PUT, y DELETE.
  - Manejo automático de JSON: Axios maneja automáticamente la serialización y deserialización de datos en formato JSON.

```vue
import axios from 'axios';

export default {
  data() {
    return {
      sensorData: null
    };
  },
  mounted() {
    axios.get('http://localhost:8000/api/sensor-data/')
      .then(response => {
        this.sensorData = response.data;
      })
      .catch(error => {
        console.error("Hubo un error:", error);
      });
  }
};
```

Código 8: Ejemplo de Uso de Axios.

Este ejemplo muestra cómo usar Axios para obtener datos del backend. En el método mounted, que se ejecuta cuando el componente es cargado, se realiza una solicitud HTTP GET para obtener los datos de los sensores [39].

- **Pinia**

  Pinia es una librería para el manejo del estado global en aplicaciones Vue 3, similar a Vuex pero diseñada específicamente para aprovechar las nuevas características de Vue 3. Pinia se utiliza para almacenar el estado global de la aplicación, como la configuración de los sensores, los valores en tiempo real y las notificaciones.

  Características Clave de Pinia:

  - Simplicidad y Flexibilidad: Pinia se enfoca en la simplicidad y usa la nueva Composition API de Vue 3 para la gestión del estado.
  - Reactividad: Pinia proporciona una forma reactiva de gestionar el estado global, lo que significa que cuando el estado cambia, las vistas asociadas se actualizan automáticamente.
  - Persistencia: Pinia soporta la persistencia del estado, lo que permite que el estado global se mantenga incluso después de una recarga de página.

```
import { defineStore } from 'pinia';

export const useSensorStore = defineStore('sensor', {
  state: () => ({
    sensorData: null,
    sensorConfig: {}
  }),
  actions: {
    setSensorData(data) {
      this.sensorData = data;
    },
    setSensorConfig(config) {
      this.sensorConfig = config;
    }
  }
});
```

Código 9: Ejemplo de Uso de Pinia.

En este ejemplo, definimos un store usando Pinia para gestionar el estado de los datos y la configuración de los sensores. Se pueden acceder a estos datos desde cualquier componente de la aplicación [38].

4.1.3.3.2. Arquitectura de la SPA

La SPA es una arquitectura de frontend que permite la creación de aplicaciones web donde la interacción del usuario no requiere recargar la página en cada acción. En lugar de eso, la aplicación carga una sola vez y actualiza dinámicamente las vistas a medida que el usuario interactúa con la interfaz. Esto resulta en una experiencia más fluida y rápida, ya que solo se cargan los datos necesarios y las vistas se actualizan sin necesidad de recargar todo el contenido de la página [51].

En este sistema de monitoreo de sensores, se ha implementado una SPA utilizando Vue 3, Vuetify, y otras tecnologías, para ofrecer una interfaz de usuario dinámica que permite interactuar con los datos de los sensores en tiempo real, configurar umbrales, y gestionar notificaciones y métricas. A continuación, describimos cómo está estructurada la SPA, sus componentes principales, y cómo se gestionan los flujos de datos entre el frontend y el backend.

- **Estructura General de la SPA**

La SPA está organizada en componentes independientes que se encargan de diferentes partes de la interfaz de usuario. Cada componente es responsable de una funcionalidad específica, lo que facilita la modularización del código y su mantenimiento [51].

Flujo de Trabajo General:

1. Carga Inicial: La aplicación se carga una sola vez, obteniendo las dependencias iniciales como Vue, Vuetify y Axios, y configurando el estado global (con Pinia) si es necesario.
2. Interacción del Usuario: A medida que el usuario interactúa con la aplicación (por ejemplo, en el Dashboard, la Configuración o las Métricas), Vue 3 maneja la reactividad y actualiza la vista de manera eficiente.
3. Consumo de la API: Las solicitudes al backend se realizan a través de Axios (para la API REST) y WebSockets (para la actualización en tiempo real de los datos). Axios gestiona las interacciones con la API RESTful, mientras que los WebSockets permiten la actualización instantánea de los datos del sensor sin necesidad de recargar la página.
4. Rendimiento: El sistema utiliza técnicas como el Lazy Loading y la carga asíncrona de componentes para optimizar el rendimiento, cargando solo los recursos necesarios cuando el usuario interactúa con ciertas partes de la aplicación.

- **Componentes Principales de la SPA**

La aplicación se organiza en varios componentes fundamentales que permiten una experiencia de usuario interactiva. A continuación, se describen los componentes clave de la SPA:

1. **Dashboard (Tiempo Real)**:
    - El componente Dashboard muestra los datos en tiempo real de los sensores. Utiliza WebSockets para recibir actualizaciones en vivo de los sensores, como la temperatura, humedad y otros parámetros.

Además, permite visualizar las alertas de los sensores si superan ciertos umbrales configurados.

- Este componente es responsable de renderizar las métricas en tiempo real y mostrar las notificaciones cuando los sensores alcanzan condiciones críticas.

2. **Configuración (Umbrales)**:
    - En el componente Configuración, los usuarios pueden establecer los umbrales mínimos y máximos para los sensores. Por ejemplo, pueden configurar los umbrales de humedad del suelo que activan la bomba de agua. Este componente interactúa con la API REST del backend para guardar y recuperar las configuraciones.
    - Además, este componente permite la gestión de la configuración general del sistema, como la elección de qué sensores monitorear o qué dispositivos controlar.
3. **Notificaciones (Log)**:
    - El componente Notificaciones muestra un registro de eventos y alertas generados por el sistema. Cuando un sensor supera un umbral configurado, se genera una notificación que aparece en este componente. Las notificaciones incluyen información sobre el sensor, el valor de la medición y la acción que se tomó, como activar o desactivar la bomba de agua.
    - Este componente también es responsable de mantener un historial de eventos, lo que permite a los usuarios revisar las acciones pasadas y los estados de los sensores.
4. **Métricas (Gráficas con ApexCharts)**:
    - El componente Métricas se encarga de mostrar gráficos interactivos de las lecturas de los sensores a lo largo del tiempo. Utiliza ApexCharts, una librería de gráficos para Vue.js, para representar

visualmente las tendencias de los datos de los sensores, como la temperatura o la humedad.

- Los gráficos permiten a los usuarios analizar el comportamiento histórico de los sensores, identificar patrones y realizar comparaciones entre diferentes periodos.

➢ **Comunicación entre Componentes y Backend**

La comunicación entre los componentes del frontend y el backend se gestiona principalmente de dos formas: a través de Axios para la API REST y WebSockets para la actualización en tiempo real.

1. API REST con Axios:
    - Axios se utiliza para hacer solicitudes a la API RESTful del backend, como obtener los datos históricos de los sensores, modificar las configuraciones de los umbrales y gestionar las notificaciones.
    - Las respuestas del backend se procesan de forma asíncrona, permitiendo que la interfaz se actualice sin bloqueos [39].

```
import axios from 'axios';

axios.get('http://localhost:8000/api/sensor-data/')
  .then(response => {
    this.sensorData = response.data; // Asignar datos a la propiedad reactiva
  })
  .catch(error => {
    console.error("Error al obtener los datos de sensores:", error);
  });
```

Código 10: Ejemplo de solicitud con Axios.

2. WebSockets para Comunicación en Tiempo Real:
    - Django Channels se encarga de gestionar la comunicación en tiempo real entre el frontend y el backend mediante WebSockets. Cuando se recibe una actualización de los sensores o una notificación de control, el backend envía los datos en tiempo real al frontend, que se encarga de actualizar la interfaz [35].

```
mounted() {
  const socket = new WebSocket('ws://localhost:8000/ws/sensor_data/');
  socket.onmessage = (event) => {
    const data = JSON.parse(event.data);
    this.sensorData = data.sensor_data; // Actualizar datos en tiempo real
  };
  socket.onclose = () => {
    console.log('Desconectado del WebSocket');
  };
}
```

Código 11: Ejemplo de implementación de WebSocket en Vue 3.

- **Ventajas de la Arquitectura SPA**

La elección de una SPA ofrece varias ventajas para el sistema:

- Interactividad Fluida: Al no necesitar recargar la página completa, las interacciones con la interfaz son más rápidas y fluidas.
- Mejor Experiencia de Usuario: El cambio entre vistas es inmediato, lo que mejora la experiencia del usuario al trabajar con los datos de los sensores.
- Optimización de Recursos: Solo se cargan los recursos necesarios cuando el usuario interactúa con ciertas funcionalidades, lo que optimiza el tiempo de carga inicial [51].

4.1.3.4. Comunicación entre Backend y Frontend

La comunicación entre el backend y el frontend se implementó mediante una combinación de API REST para el intercambio de información estructurada y WebSockets para la transmisión en tiempo real.

El backend actuó como servidor de datos, enviando actualizaciones periódicas al frontend cada vez que se registraban nuevas lecturas o se generaban predicciones. Esta sincronización permitió mantener la interfaz siempre actualizada y garantizó la consistencia entre las acciones del usuario y el estado del sistema.

La arquitectura adoptada favoreció una operación continua y estable, donde la capa web funcionó como puente entre el entorno físico (sensores y actuadores) y el entorno lógico (modelo predictivo y control del sistema). Gracias a esta integración,

el sistema de riego inteligente pudo ofrecer monitoreo, análisis y actuación en una misma plataforma.

4.1.3.4.1. API REST con Axios

La API REST se utiliza para gestionar las solicitudes más tradicionales entre el frontend y el backend. Permite al frontend obtener información de manera síncrona, como la configuración de los sensores, los datos históricos o los registros de las notificaciones.

Características Clave de la API REST:

- Petición Síncrona: El frontend realiza peticiones HTTP (usualmente GET o POST) al backend para obtener o modificar los datos.
- Interacción con el Backend: Axios es la librería utilizada en el frontend para interactuar con la API, realizando las peticiones de manera asíncrona.

```
import axios from 'axios';

export default {
  data() {
    return {
      sensorData: [],
      loading: false,
    };
  },
  mounted() {
    this.fetchSensorData();
  },
  methods: {
    fetchSensorData() {
      this.loading = true;
      axios.get('http://localhost:8000/api/sensor-data/')
        .then(response => {
          this.sensorData = response.data;  // Asignar los datos obtenidos a una
propiedad reactiva
        })
        .catch(error => {
          console.error('Error fetching sensor data:', error);
        })
        .finally(() => {
          this.loading = false;
        });
    },
  },
};
```

Código 12: Ejemplo de uso de Axios para obtener datos de la API.

Explicación:

- En este ejemplo, el método **`fetchSensorData`** realiza una solicitud GET a la API para obtener los datos de los sensores.
- Los datos obtenidos se asignan a la propiedad reactiva **`sensorData`**, que automáticamente actualiza la vista de la aplicación.
- **`axios.get`** realiza la llamada al backend y la respuesta se maneja de manera asíncrona, actualizando la interfaz de usuario cuando los datos están disponibles [39].

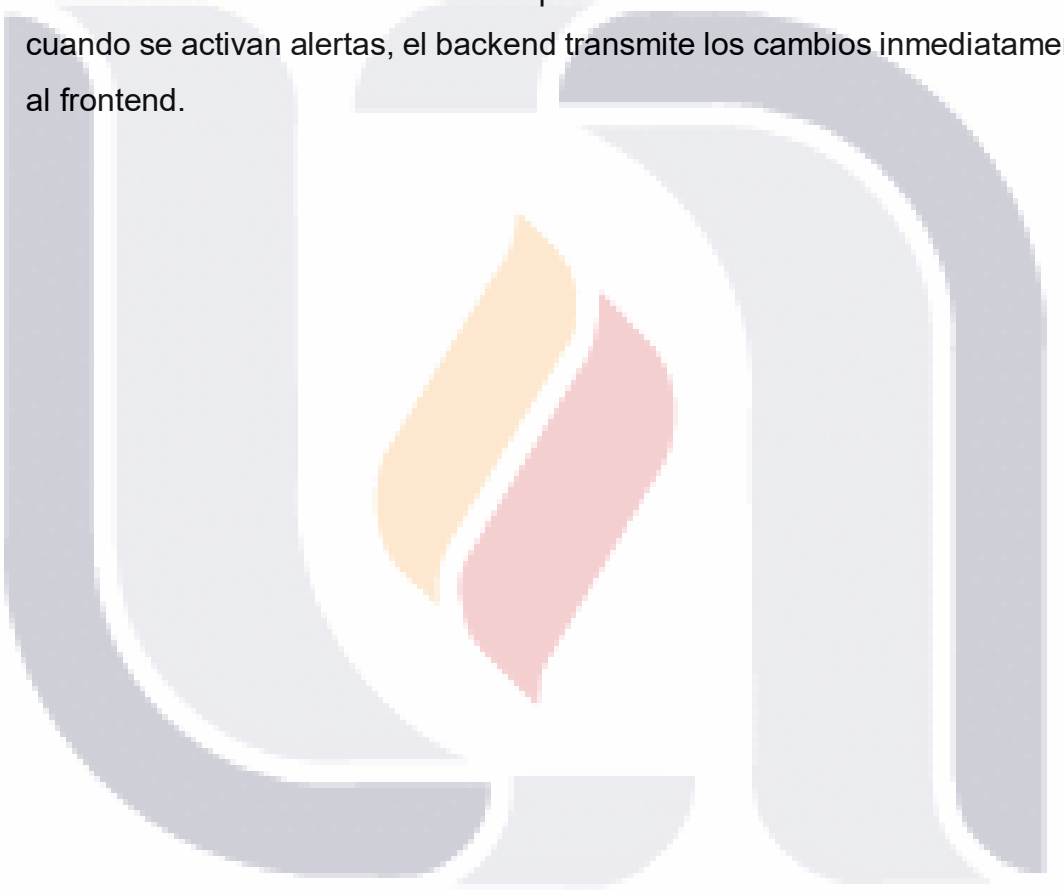
4.1.3.4.2. WebSockets para Comunicación en Tiempo Real

WebSockets permiten la comunicación bidireccional en tiempo real entre el frontend y el backend. En este sistema, Django Channels se utiliza para gestionar las

conexiones WebSocket y permitir que el backend envíe actualizaciones en tiempo real al frontend, como nuevos datos de los sensores o notificaciones.

Características Clave de WebSockets:

- Comunicación Bidireccional: Los WebSockets permiten que el backend envíe actualizaciones al frontend en tiempo real, sin necesidad de que el cliente realice una solicitud.
- Actualizaciones en Vivo: A medida que los sensores envían nuevos datos o cuando se activan alertas, el backend transmite los cambios inmediatamente al frontend.



```vue
<template>
  <div>
    <h1>Dashboard en Tiempo Real</h1>
    <div v-if="loading">Cargando datos...</div>
    <div v-else>
      <p>Última medición: {{ sensorData.temperature }} °C</p>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      socket: null,
      sensorData: {},
      loading: true,
    };
  },
  mounted() {
    this.connectWebSocket();
  },
  methods: {
    connectWebSocket() {
      this.socket = new WebSocket('ws://localhost:8000/ws/sensor_data/');

      this.socket.onmessage = (event) => {
        const data = JSON.parse(event.data);
        this.sensorData = data.sensor_data; // Actualizamos los datos del sens
        this.loading = false;
      };

      this.socket.onclose = () => {
        console.log('Desconectado del WebSocket');
      };
    }
  },
};
</script>
```

Código 13: Ejemplo de configuración de WebSocket en Vue 3.

Explicación:

- En este ejemplo, se establece una conexión WebSocket con el backend utilizando la URL proporcionada (**`ws://localhost:8000/ws/sensor_data/`**).
- Cuando el servidor envía un mensaje (por ejemplo, una actualización de datos de sensores), el cliente (frontend) recibe el mensaje en tiempo real y actualiza la interfaz sin necesidad de recargar la página.

- **`onmessage`** gestiona la actualización de los datos cada vez que se recibe una nueva información del backend [32], [35].

Beneficios de WebSockets en el Sistema:

- Actualizaciones en tiempo real: Los datos se actualizan en la interfaz de usuario a medida que cambian en el backend, sin necesidad de recargar la página o realizar peticiones constantes.
- Reducción de Carga de Servidor: Dado que los WebSockets mantienen una conexión persistente, no es necesario hacer múltiples solicitudes HTTP, lo que reduce la carga en el servidor.
- Interactividad Dinámica: Los usuarios reciben información instantánea de los sensores y las notificaciones, mejorando la experiencia de uso.

4.1.3.4.3. Integración entre WebSockets y API REST

La combinación de WebSockets y API REST permite gestionar de manera eficiente tanto las operaciones de consulta y configuración como las actualizaciones en tiempo real. La API REST se utiliza para acceder y modificar los datos de manera síncrona (por ejemplo, obtener configuraciones o datos históricos), mientras que los WebSockets permiten la transmisión instantánea de nuevos datos o alertas.

Flujo de Datos:

1. Inicio de la aplicación: Al cargar la SPA, el frontend realiza solicitudes REST a través de Axios para obtener la configuración inicial y los datos históricos de los sensores.
2. Conexión a WebSocket: Una vez que la aplicación está cargada, el frontend establece una conexión WebSocket con el backend para recibir las actualizaciones en tiempo real de los sensores.
3. Recepción de datos: A medida que los sensores envían nuevas lecturas o se activan notificaciones, el backend envía estos datos al frontend a través de WebSockets, que los actualiza de inmediato en la interfaz.

4.1.3.4.4. Componentes Principales

La SPA está compuesta por varios componentes principales, que forman la interfaz de usuario interactiva y permiten a los usuarios visualizar los datos de los sensores en tiempo real, configurar los umbrales, revisar las notificaciones y analizar las métricas históricas de los sensores. Cada uno de estos componentes es responsable de una parte específica del sistema, y su interacción con el backend se maneja mediante WebSockets y API REST.

- **Dashboard**

El Dashboard es el componente principal donde los usuarios pueden visualizar los datos de los sensores en tiempo real. Este componente se encarga de mostrar información actualizada constantemente sobre las lecturas de los sensores (como la temperatura, humedad, y otros parámetros) y cualquier alerta generada, todo ello sin necesidad de recargar la página.

Funcionalidad:

- Actualización en tiempo real: Utiliza WebSockets para recibir las lecturas de los sensores a medida que se generan en el backend. Esto permite que los usuarios vean las mediciones más recientes sin necesidad de refrescar la página.
- Visualización de datos clave: Muestra las métricas principales, como la temperatura, humedad y otros parámetros medidos por los sensores, en un formato claro y accesible.
- Notificaciones: Si los valores de los sensores superan los umbrales configurados, el Dashboard muestra notificaciones visuales, como cambios de color o mensajes de alerta.

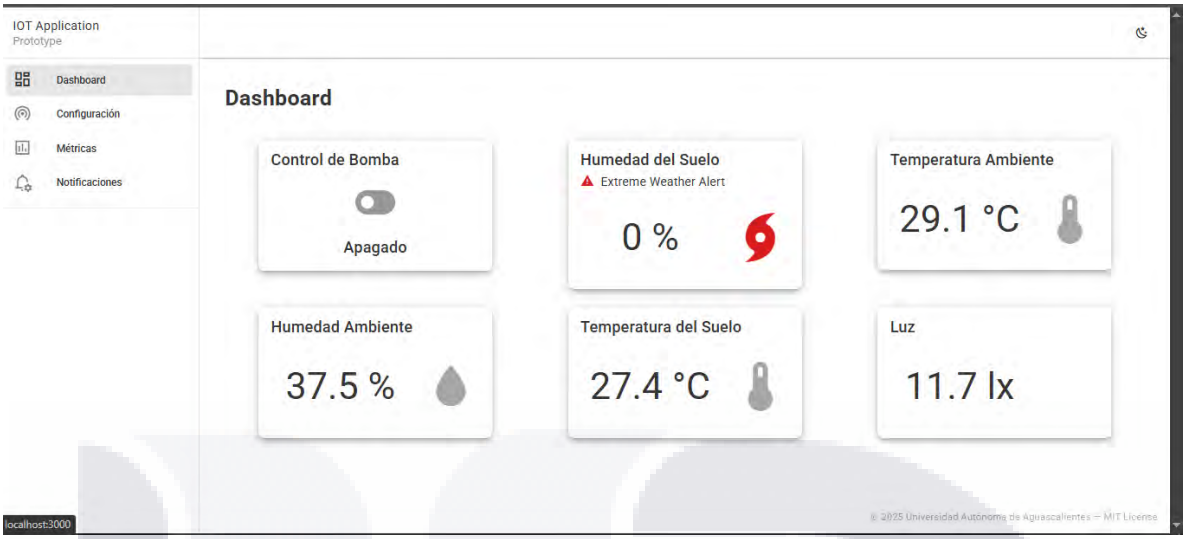


Imagen 2. Captura de pantalla de Dashboard.

➢ Configuración

El componente Configuración permite a los usuarios establecer los umbrales para los sensores. Estos umbrales pueden ser configurados para controlar cuando ciertos parámetros del sistema (como la humedad o la temperatura) activan o desactivan ciertos dispositivos, como la bomba de agua. Los usuarios también pueden visualizar y ajustar los valores mínimos y máximos para cada sensor.

Funcionalidad:

- Ajuste de umbrales: Los usuarios pueden ingresar valores para los umbrales mínimos y máximos de los sensores.
- Interacción con la API REST: Los cambios en la configuración de los sensores se envían al backend a través de una API REST para almacenarlos en la base de datos.

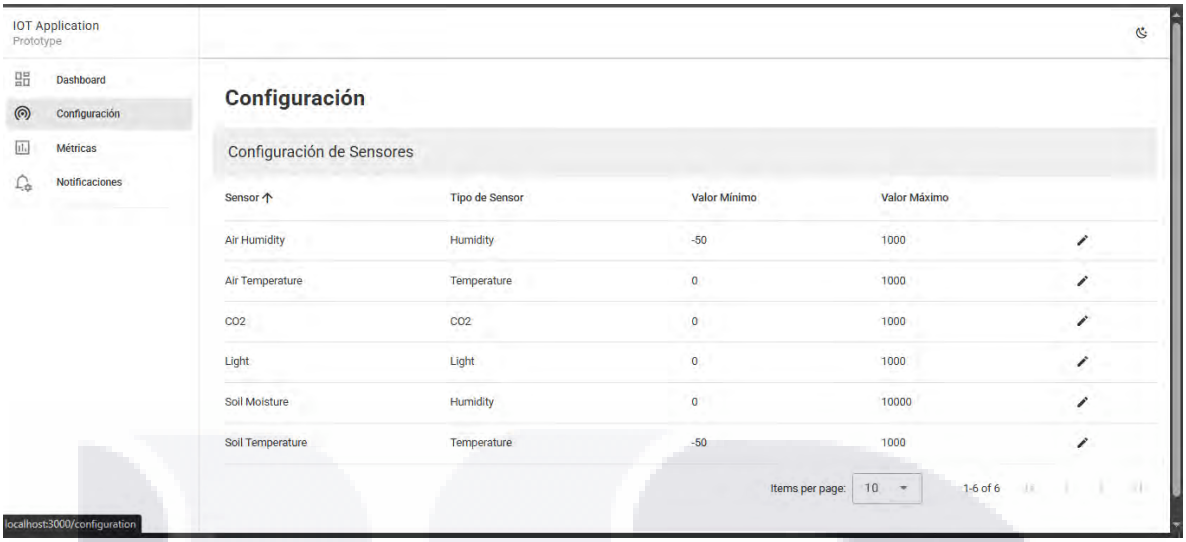


Imagen 3. Captura de pantalla de Configuración.

Imagen 4. Captura de pantalla de Editar Sensor.

- **Notificaciones**

El componente Notificaciones muestra un registro de todas las alertas y eventos importantes generados por el sistema. Este componente es esencial para que los usuarios sigan de cerca los cambios y las intervenciones del sistema, como los

valores de los sensores que superan los umbrales predefinidos o las acciones que se han tomado (como activar la bomba de agua).

Funcionalidad:

- Registro de alertas: Muestra un log de todas las notificaciones generadas, con detalles sobre el sensor, el tipo de alerta y la acción tomada.
- Interacción con la API REST: Las notificaciones se pueden almacenar y recuperar a través de la API REST.

Imagen 5. Captura de pantalla de Notificaciones.

➢ Métricas

El componente Métricas utiliza ApexCharts para mostrar gráficos interactivos de los datos de los sensores a lo largo del tiempo. Este componente es útil para analizar el comportamiento histórico de los sensores y facilitar la toma de decisiones, como la optimización de los umbrales de los sensores.

Funcionalidad:

- Visualización de datos históricos: Utiliza ApexCharts para crear gráficos de líneas, barras, o áreas que muestran los datos históricos de los sensores, como la evolución de la temperatura o humedad a lo largo del tiempo.

- Interactividad: Los gráficos permiten al usuario interactuar con los datos, ver detalles y realizar comparaciones entre diferentes períodos.

Imagen 6. Captura de pantalla de Métricas.

4.1.3.5. Flujo de Datos Integrado

El flujo de datos en este sistema de monitoreo de sensores está diseñado para ser eficiente y en tiempo real. El objetivo es capturar los datos de los sensores, procesarlos, almacenarlos de forma adecuada en la base de datos y finalmente visualizarlos para el usuario de manera interactiva y en tiempo real. A continuación, describimos detalladamente cómo se gestionan estos datos a lo largo de su recorrido, desde la adquisición en los sensores hasta su visualización en el frontend.

4.1.3.5.1. Adquisición de Datos desde los Sensores

El primer paso en el flujo de datos es la adquisición de datos desde los sensores instalados en el sistema de monitoreo. Estos sensores miden varios parámetros del entorno, como temperatura, humedad, niveles de CO2, entre otros.

Protocolo MQTT:

1. Conexión al bróker HiveMQ: Los sensores envían sus datos al backend a través del protocolo MQTT. El backend se conecta a un bróker HiveMQ, que recibe y distribuye los datos enviados por los sensores.

2. Publicación de datos: Los sensores publican los datos en temas específicos de MQTT. El backend se suscribe a estos temas y recibe los datos a medida que son publicados.

Flujo:

- Sensores → Bróker MQTT → Backend.

4.1.3.5.2. Procesamiento y Almacenamiento de Datos

Una vez que los datos son recibidos por el backend a través de MQTT, el siguiente paso es procesarlos y almacenarlos de forma eficiente en la base de datos. El sistema utiliza TimescaleDB para almacenar los datos de los sensores, aprovechando sus características de optimización para datos temporales.

Procesamiento de Datos:

- Los datos recibidos (por ejemplo, lecturas de temperatura o humedad) se procesan para verificar su validez y determinar si requieren alguna acción, como generar una alerta.
- Los datos se validan para asegurarse de que estén dentro de los rangos aceptables, y si no lo están, se generan notificaciones o alertas.

Almacenamiento en TimescaleDB:

- Los datos procesados se almacenan en una tabla de `sensor_data` dentro de TimescaleDB, que permite realizar consultas rápidas y eficientes sobre datos de series temporales.

Flujo:

- Backend → Tarea de Celery → Base de datos (TimescaleDB): Los datos son almacenados en la base de datos para su posterior consulta y análisis.

4.1.3.5.3. Visualización en Tiempo Real (Frontend)

Una vez que los datos han sido almacenados en la base de datos, el siguiente paso es la visualización en tiempo real de los mismos para el usuario. Este flujo involucra

la actualización constante de los datos en el frontend a medida que los sensores envían nuevas mediciones.

WebSockets para Actualización en Tiempo Real:

- El frontend se conecta al backend a través de WebSockets, lo que permite que el backend envíe actualizaciones en tiempo real a medida que los datos de los sensores cambian.
- El frontend, por ejemplo, en el Dashboard, recibe los datos y actualiza los valores mostrados sin necesidad de recargar la página.

Flujo:

- Backend (WebSocket) → Frontend (Vue 3): Los datos de los sensores se envían al frontend en tiempo real.

4.1.3.5.4. Visualización de Datos Históricos (Frontend)

Además de la visualización en tiempo real, es crucial poder consultar y analizar los datos históricos. El componente Métricas en el frontend permite a los usuarios visualizar gráficas de los datos almacenados en la base de datos a lo largo del tiempo.

Consultas a la API REST:

- El frontend realiza consultas a la API REST para obtener los datos históricos almacenados en TimescaleDB.
- Estos datos se visualizan utilizando gráficos interactivos con ApexCharts.

Flujo:

- Frontend (Axios) → Backend (API REST): Los datos históricos se obtienen a través de peticiones REST y se visualizan en gráficos.

4.1.3.5.5. Notificaciones y Alertas (Frontend)

En el caso de que los sensores detecten un valor fuera de los umbrales predefinidos, el sistema debe generar notificaciones para informar a los usuarios. Estas notificaciones se muestran en el componente Notificaciones del frontend.

Generación de Notificaciones:

- Las notificaciones son generadas en el backend cuando se detectan valores fuera de los umbrales configurados. Estas alertas se envían a los usuarios a través de WebSockets y se visualizan en el frontend.

Flujo:

- Backend (WebSocket) → Frontend (Vue 3 - Notificaciones): Las alertas y notificaciones se envían en tiempo real al frontend.

4.1.3.5.6. Resumen del Flujo de Datos

1. Adquisición de Datos: Los sensores publican datos al backend a través de MQTT. El backend se suscribe a estos temas y recibe los datos en tiempo real.
2. Procesamiento de Datos: El backend procesa los datos recibidos, los valida y los almacena en TimescaleDB.
3. Visualización en Tiempo Real: El frontend utiliza WebSockets para recibir actualizaciones en tiempo real y mostrar los datos en el Dashboard.
4. Consulta de Datos Históricos: Los datos históricos se obtienen a través de la API REST y se visualizan utilizando ApexCharts.
5. Notificaciones: Las alertas se envían desde el backend al frontend mediante WebSockets, permitiendo que los usuarios reciban notificaciones inmediatas.

4.1.3.6. Seguridad y Consideraciones Técnicas

El sistema de monitoreo de sensores debe cumplir con varios requisitos técnicos que aseguren su funcionamiento correcto, seguro y eficiente. Para ello, se han implementado diversas prácticas de seguridad, validación de datos, control de

errores, y optimización de rendimiento y escalabilidad. A continuación, se detallan cada una de estas consideraciones.

4.1.3.6.1. Seguridad

La seguridad es una de las principales preocupaciones en cualquier sistema conectado a redes, especialmente cuando se maneja información sensible como los datos de los sensores y configuraciones del sistema. Las prácticas implementadas para asegurar la protección de los datos y la integridad del sistema incluyen:

- **Autenticación y Autorización**

El sistema emplea mecanismos robustos de autenticación y autorización para garantizar que solo los usuarios legítimos puedan acceder y modificar los datos o la configuración del sistema.

- Autenticación con JWT (JSON Web Tokens): Se ha implementado JWT para asegurar que solo los usuarios autenticados puedan interactuar con la API REST. El token JWT se genera al momento del inicio de sesión y debe incluirse en cada solicitud de la API para verificar que el usuario tiene permisos adecuados.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
}
```

Código 14: Ejemplo de configuración en Django REST Framework.

- Autorización basada en roles: Dependiendo del tipo de usuario (por ejemplo, administrador o usuario normal), se han implementado controles de acceso a ciertos recursos. Los usuarios con mayor nivel de privilegios pueden configurar los umbrales de los sensores y modificar la configuración del sistema [31].

- **Cifrado de Datos**

La transmisión de datos entre el frontend y el backend se realiza utilizando HTTPS para cifrar la información durante su transmisión y protegerla de posibles ataques como Man-in-the-Middle. Además, los tokens de autenticación (JWT) también están cifrados para garantizar su seguridad.

- Uso de HTTPS: Se ha configurado el backend para obligar a todas las comunicaciones a ser realizadas a través de HTTPS.
- Cifrado de datos sensibles: Los datos sensibles, como las configuraciones de los sensores o los tokens de autenticación, se cifran y se almacenan de forma segura [31].

➢ **Prevención de Ataques Comunes**

- Protección contra ataques de CSRF (Cross-Site Request Forgery): Django ya incluye protección integrada contra ataques CSRF. Para las vistas que utilizan formularios, se verifica que las solicitudes provengan de una fuente confiable.
- Protección contra XSS (Cross-Site Scripting): Se asegura que los datos proporcionados por el usuario se saniticen adecuadamente antes de ser mostrados en la interfaz para evitar la ejecución de código malicioso [31].

4.1.3.6.2. Validaciones de Datos

La validación de los datos es fundamental para garantizar que solo se procesen datos correctos y dentro de los rangos esperados. Se implementan múltiples capas de validación tanto en el frontend como en el backend:

➢ **Validación en el Backend**

En el backend, se utilizan validaciones para asegurarse de que los datos provenientes de los sensores, como la temperatura y la humedad, sean válidos antes de almacenarlos en la base de datos.

- Validación de rangos: Se comprueba que los valores de los sensores se encuentren dentro de los límites definidos, como la temperatura entre -10°C y 50°C, o la humedad entre 0% y 100%.

```python
from django.core.exceptions import ValidationError

def validate_sensor_data(data):
    if data['temperature'] < -10 or data['temperature'] > 50:
        raise ValidationError("La temperatura debe estar entre -10°C y 50°C.")
    if data['humidity'] < 0 or data['humidity'] > 100:
        raise ValidationError("La humedad debe estar entre 0 y 100.")
    return True
```

Código 15: Ejemplo de validación en Django.

- **Validación en el Frontend**

En el frontend, antes de enviar los datos del formulario de configuración de sensores, se valida que los valores introducidos por el usuario sean correctos y dentro de los rangos aceptables.

- Validación de campos de entrada: Aseguramos que los valores de configuración, como los umbrales de los sensores, sean números dentro de los rangos permitidos antes de enviar la solicitud al backend.

4.1.3.6.3. Control de Errores

El manejo adecuado de errores es esencial para garantizar que el sistema siga funcionando de manera estable incluso cuando se presenten problemas. En este sistema, se implementan varias estrategias para capturar, registrar y manejar errores.

- **Control de Errores en el Backend**

En el backend, se gestionan posibles errores de conexión, errores al procesar datos y excepciones generales. Cuando ocurre un error, se registra en el sistema para su posterior análisis.

- Manejo de excepciones generales: Las excepciones se capturan y se devuelven mensajes adecuados a los usuarios, sin exponer detalles técnicos innecesarios.

```python
from rest_framework.exceptions import APIException

class CustomAPIException(APIException):
    status_code = 500
    default_detail = 'Se ha producido un error interno en el servidor.'
    default_code = 'error'

# Uso de la excepción personalizada en vistas
try:
    # Lógica del backend
except Exception as e:
    raise CustomAPIException(detail=str(e))
```

Código 16: Ejemplo de manejo de errores en Django.

- **Control de Errores en el Frontend**

En el frontend, se implementan manejadores de errores para capturar posibles fallos al realizar solicitudes HTTP o WebSocket, y proporcionar retroalimentación al usuario.

Manejo de errores en Axios:

```javascript
axios.get('http://localhost:8000/api/sensor-data/')
  .then(response => {
    this.sensorData = response.data;
  })
  .catch(error => {
    console.error('Error al obtener los datos:', error);
    this.errorMessage = "Hubo un problema al obtener los datos del sensor.";
  });
```

Código 17: Ejemplo de manejo de errores con Axios.

4.1.3.6.4. Rendimiento

Para garantizar que el sistema sea eficiente, se han implementado prácticas para optimizar el rendimiento tanto en el backend como en el frontend.

- **Optimización en el Backend**

- Consultas eficientes: Se han optimizado las consultas SQL a la base de datos TimescaleDB para asegurar que las operaciones de lectura y escritura sean rápidas y eficientes, especialmente cuando se manejan grandes volúmenes de datos.

- Uso de índices: En las tablas clave, se utilizan índices para acelerar las consultas y mejorar el rendimiento general.

➢ **Optimización en el Frontend**

- Lazy Loading: Se implementa lazy loading para cargar los componentes solo cuando sean necesarios, lo que mejora los tiempos de carga inicial de la aplicación.
- Minificación de recursos: Los recursos estáticos como JavaScript y CSS se minifican para reducir el tamaño de la página y mejorar los tiempos de carga.

4.1.3.6.5. Escalabilidad

La escalabilidad es una consideración importante en sistemas que deben manejar grandes volúmenes de datos o que pueden crecer con el tiempo. Se han implementado prácticas para asegurar que el sistema pueda escalar fácilmente a medida que aumenten las demandas de los usuarios.

➢ **Escalabilidad en el Backend**

- Uso de Redis: Redis se utiliza como broker de Celery para la gestión de tareas asíncronas. Redis es un sistema de almacenamiento en memoria que permite manejar grandes volúmenes de tareas sin afectar el rendimiento del sistema.
- Base de datos escalable: TimescaleDB está diseñado para gestionar grandes volúmenes de datos de series temporales. Su capacidad para particionar datos en hypertables y usar políticas de retención de datos ayuda a mantener el rendimiento a medida que aumenta la cantidad de datos.

➢ **Escalabilidad en el Frontend**

- Uso de componentes modulares: La SPA está construida con componentes modulares, lo que facilita la incorporación de nuevas funcionalidades sin afectar el rendimiento general de la aplicación.

- Optimización de recursos estáticos: Se implementa la carga asíncrona de recursos estáticos, lo que permite que solo se carguen los elementos necesarios según las interacciones del usuario.

4.1.4. Subsistema Predictivo

El componente de inteligencia artificial del sistema de riego inteligente se basa en la implementación de un modelo de aprendizaje supervisado, cuyo objetivo es predecir el nivel futuro de humedad del suelo y, a partir de ello, sugerir de forma anticipada si se requiere activar el riego [52], [53]. Esta predicción permite tomar decisiones basadas en patrones temporales y ambientales en lugar de respuestas instantáneas, mejorando la eficiencia hídrica y previniendo tanto el riego excesivo como el estrés hídrico del cultivo [54].

4.1.4.1. Datos utilizados para el entrenamiento

El entrenamiento del modelo de machine learning propuesto en este proyecto se basa en un conjunto de datos estructurado que simula las condiciones ambientales y del suelo en un entorno agrícola protegido, como un invernadero. Este conjunto de datos se diseñó con el propósito de proporcionar al modelo información suficiente y representativa que le permitiera identificar patrones y relaciones temporales entre las variables del entorno y los cambios en la humedad del suelo, la cual es la variable objetivo a predecir [52], [53].

Dado que la implementación en campo real no fue posible durante el desarrollo del proyecto, se optó por generar un conjunto de datos sintéticos simulados, apoyados en rangos y comportamientos obtenidos de fuentes confiables, como artículos científicos, tesis previas, manuales técnicos agrícolas, y literatura especializada en microclimas agrícolas de invernaderos en zonas semiáridas, como Aguascalientes [53].

4.1.4.1.1. Características generales del conjunto de datos

El conjunto de datos simulado contiene aproximadamente 2,000 registros cronológicos, generados en intervalos regulares de 15 minutos, abarcando un período virtual de dos semanas continuas. Cada registro representa una captura simultánea de múltiples variables sensadas en el entorno del cultivo, y se estructura como una fila de entrada en una base de datos de series de tiempo, replicando el comportamiento que tendría una base de datos real bajo condiciones operativas del sistema [52].

Las variables consideradas y registradas en cada observación son las siguientes:

Variable	Tipo	Unidad	Descripción
`soil_moisture`	Continua	Porcentaje (%)	Humedad del suelo, objetivo del modelo.
`soil_temperature`	Continua	Grados Celsius (°C)	Temperatura del suelo.
`air_temperature`	Continua	Grados Celsius (°C)	Temperatura ambiental.
`humidity`	Continua	Porcentaje (%)	Humedad relativa del aire.
`light_intensity`	Continua	Lux	Intensidad de luz ambiental.
`co2_concentration`	Continua	ppm	Concentración de dióxido de carbono.
`hour`	Discreta	Horas (0–23)	Hora del día, representada como variable numérica.
`delta_minutes`	Discreta	Minutos	Intervalo de tiempo transcurrido desde el último registro.

Tabla 1. Variables usadas para entrenar el modelo.

La variable objetivo (`soil_moisture`) representa la humedad del suelo que el sistema busca predecir, en tanto que las demás variables actúan como entradas del modelo (features) que ayudan a estimar su comportamiento futuro [55], [56].

4.1.4.1.2. Justificación de las variables utilizadas

- Temperatura del suelo y del aire: influye directamente en la tasa de evaporación y en la demanda hídrica de la planta.
- Humedad relativa ambiental: afecta la pérdida de agua por transpiración y evapotranspiración.
- Intensidad de luz: correlacionada con la actividad fotosintética y el cierre/apertura de estomas, lo cual modifica el consumo de agua por parte del cultivo.
- Concentración de CO_2: considerada en estudios recientes como un factor que puede alterar la fisiología del cultivo y su eficiencia hídrica.
- Hora del día: integra un componente temporal cíclico importante, pues el riego depende en gran medida de la radiación solar y temperatura ambiente, las cuales siguen ciclos diarios.
- Intervalo temporal (delta_minutes): permite que el modelo tenga noción del ritmo de cambio entre registros, útil para representar pendiente de descenso de humedad.

Estas variables, combinadas, permiten al modelo aprender las relaciones no lineales y temporales que ocurren en un entorno agrícola real y que afectan la retención o pérdida de humedad en el sustrato [55], [56], [57].

4.1.4.1.3. Simulación de condiciones reales

Para garantizar que los datos simulados fueran representativos, se definieron rangos realistas para cada variable, con base en condiciones típicas observadas en invernaderos [52]:

Variable	Rango simulado
Humedad del suelo	25 % – 80 %
Temperatura del suelo	16 °C – 30 °C
Temperatura ambiental	18 °C – 38 °C
Humedad relativa	40 % – 95 %
Luz ambiental	100 – 45,000 lux
CO_2	380 – 800 ppm

Tabla 2. Rangos de variables usados en la simulación.

Se definieron perfiles diarios que simulan días soleados, nublados, con ventilación natural o forzada, incluyendo también intervalos con y sin activación del riego para reflejar saltos o estabilización en la humedad del suelo [53].

Los registros se almacenaron en formato .csv y posteriormente se importaron a una base de datos TimescaleDB, la cual fue utilizada tanto para almacenar los datos de entrenamiento como para representar en forma realista el comportamiento del sistema completo en la etapa de simulación [58].

4.1.4.1.4. Ventajas del enfoque simulado

Aunque trabajar con datos simulados implica limitaciones, este enfoque permitió:

- Controlar las condiciones experimentales y probar distintos escenarios en menor tiempo.
- Tener una base de datos limpia, sin ruido, para probar la arquitectura del modelo.
- Validar el funcionamiento general del sistema, desde la predicción hasta la decisión de riego, sin depender de factores logísticos o meteorológicos reales.

- Entrenar el modelo sin riesgo operativo, ya que no se ejecutan acciones físicas sobre un cultivo real [52], [58].

4.1.4.1.5. Limitaciones

- La relación entre variables puede no reflejar con total precisión las condiciones microclimáticas de un cultivo protegido real.
- No se incluyeron factores estacionales, ni efectos acumulativos de riegos anteriores o lluvias.
- No fue posible incorporar ruido sensorial, interferencias o fallos comunes en sensores reales, lo que puede afectar el comportamiento del modelo al aplicarlo en campo [53].

4.1.4.2. Proceso de entrenamiento

Una vez generado y estructurado el conjunto de datos, se procedió al diseño y ejecución del proceso de entrenamiento del modelo de *machine learning*, cuya finalidad es aprender la relación entre las condiciones ambientales y la evolución de la humedad del suelo a lo largo del tiempo. Esta sección describe en detalle las etapas involucradas en el desarrollo del modelo predictivo: desde el preprocesamiento hasta la validación y evaluación de desempeño.

4.1.4.2.1. Herramientas utilizadas

El entrenamiento del modelo fue realizado utilizando el lenguaje de programación Python, dada su versatilidad y amplio ecosistema de bibliotecas científicas. Las principales herramientas empleadas fueron:

- Pandas: para manipulación, filtrado y análisis de datos tabulares.
- NumPy: para operaciones vectorizadas y estructuras matriciales.
- Matplotlib / Seaborn: para visualización de distribución de datos y curvas de error.
- Scikit-learn: biblioteca principal utilizada para el desarrollo del modelo de regresión, así como para su validación, análisis de errores y exportación.

Todo el proceso se desarrolló en un entorno virtual controlado, utilizando Jupyter Notebooks como interfaz interactiva de trabajo y documentación del código [59].

4.1.4.2.2. Preprocesamiento de los datos

Antes de alimentar los datos al modelo, se llevaron a cabo diversas operaciones de preparación con el fin de garantizar la calidad de los datos de entrada y evitar sesgos:

1. Limpieza de datos:
    - Se eliminaron registros con valores nulos o inconsistentes (por ejemplo, humedad del suelo mayor al 100 %).
    - Se aplicaron filtros para descartar valores atípicos generados en la simulación (e.g., temperatura ambiente < 0 °C).
2. Codificación de variables temporales:
    - La variable "hora del día" se codificó como un número entero de 0 a 23, permitiendo que el modelo reconozca los patrones circadianos del entorno agrícola.
3. Normalización de variables:
    - Se aplicó escalado min-max a todas las variables predictoras (entre 0 y 1), con el objetivo de evitar que las diferencias de escala entre variables (por ejemplo, CO_2 en ppm vs. humedad en %) afectaran negativamente el entrenamiento del modelo [59], [60].
4. Separación de conjuntos:
    - Se dividió el conjunto total de datos en dos subconjuntos:
        - Entrenamiento (80 %): utilizado para construir el modelo.
        - Prueba (20 %): utilizado para evaluar el modelo con datos no vistos.

4.1.4.2.3. Selección del algoritmo

Con base en la naturaleza del problema —predicción de una variable continua (humedad del suelo)— se seleccionó un enfoque de regresión supervisada. Se exploraron varios algoritmos disponibles en Scikit-learn para identificar el que ofreciera el mejor equilibrio entre precisión, bajo sobreajuste y facilidad de interpretación [55], [56]:

- Regresión lineal múltiple: como punto de partida base.
- Árboles de decisión (DecisionTreeRegressor): por su capacidad de capturar relaciones no lineales.
- k-Nearest Neighbors (KNN Regressor): para comparación con métodos basados en vecindarios.
- Random Forest Regressor *(etapa exploratoria)*: se consideró, aunque no fue el elegido por su mayor costo computacional.

Tras evaluar los modelos, se determinó que el árbol de decisión proporcionaba un mejor ajuste a los patrones simulados, sin incurrir en sobreajuste, y con una interpretabilidad alta que lo hacía adecuado para una primera implementación funcional.

4.1.4.2.4. Entrenamiento del modelo

Con el algoritmo seleccionado, se procedió a entrenar el modelo con el conjunto de entrenamiento. Durante esta etapa se llevaron a cabo los siguientes pasos:

- Entrenamiento inicial con los parámetros por defecto del algoritmo.
- Análisis de importancia de variables: utilizando el atributo `feature_importances_`, se identificó que las variables más influyentes fueron [59]:
    - Humedad relativa
    - Temperatura del suelo
    - Hora del día

- Luz ambiental

- Evaluación inicial de desempeño:
  - Se utilizaron métricas estándar:
    - MAE (Mean Absolute Error)
    - MSE (Mean Squared Error)
    - R^2 (Coeficiente de determinación)
  - Los resultados preliminares fueron:
    - MAE: 2.85 %
    - RMSE: 4.17 %
    - R^2: 0.91
- Ajuste de hiperparámetros (tuning):
  - Se aplicó búsqueda de cuadrícula (GridSearchCV) para ajustar:
    - Profundidad máxima del árbol (**`max_depth`**)
    - Mínimo de muestras por nodo (**`min_samples_split`**)
    - Mínimo de muestras por hoja (**`min_samples_leaf`**)
  - El modelo ajustado presentó una mejora marginal en MSE sin aumentar la complejidad del árbol [59], [60].

4.1.4.2.5. Validación cruzada

Para garantizar que el modelo no estuviera sobreajustado a los datos de entrenamiento, se aplicó validación cruzada con $k = 5$ [60], es decir, el conjunto de datos se dividió en cinco partes, entrenando el modelo con cuatro y validando con la quinta en cada iteración.

Los resultados de la validación cruzada fueron consistentes, lo que sugiere que el modelo generaliza adecuadamente y no depende en exceso de subconjuntos particulares de datos [59], [60].

Métrica	Valor promedio
MAE	2.97
RMSE	4.26
R^2	0.89

Tabla 3. Resultados de la validación cruzada.

4.1.4.2.6. Exportación del modelo

Tras el entrenamiento, el modelo final fue serializado utilizando la librería **`joblib`**, lo que permitió guardar su estructura y parámetros entrenados en un archivo binario (`.pkl`). Esta serialización facilitó su posterior integración en el backend del sistema, donde puede ser cargado dinámicamente sin requerir nuevo entrenamiento cada vez que el servidor se reinicia [52], [59].

```
from joblib import dump, load
dump(trained_model, 'humidity_predictor_model.pkl')
```

Código 18. Serialización del modelo entrenado con joblib en formato .pkl.

El modelo entrenado quedó listo para ser utilizado por el backend como un servicio predictivo que toma datos recientes del entorno y devuelve una estimación de la humedad futura del suelo.

4.1.4.3. Toma de decisiones basada en predicción

Una vez entrenado y validado el modelo de *machine learning*, su integración al sistema de riego inteligente tiene como propósito principal asistir en la toma automatizada o semi-automatizada de decisiones relacionadas con la activación del riego, basándose en la predicción futura del nivel de humedad del suelo. Esta capacidad predictiva permite que el sistema anticipe necesidades hídricas del

cultivo antes de que se presenten condiciones críticas, aportando valor en términos de eficiencia, sostenibilidad y autonomía operativa [52], [53].

El modelo predictivo no actúa de forma aislada, sino que se incorpora como una unidad funcional dentro del backend, conectada a un motor de reglas que define el comportamiento del sistema en función de los resultados del modelo y las condiciones establecidas por el usuario.

4.1.4.3.1. Lógica de operación del sistema predictivo

El modelo predictivo se ejecuta de forma periódica o en respuesta a eventos del sistema, como la llegada de nuevos datos sensoriales desde la Raspberry Pi. El flujo general de decisión se describe a continuación:

1. Recepción de datos actuales: el backend recibe un nuevo conjunto de datos desde los sensores conectados a la Raspberry Pi, incluyendo:
    - Humedad actual del suelo
    - Temperatura del suelo y ambiente
    - Humedad relativa
    - Intensidad de luz ambiental
    - Nivel de CO_2
    - Hora del día
2. Preparación de datos: los datos entrantes se normalizan y ordenan para que coincidan con el formato de entrada requerido por el modelo serializado.
3. Ejecución del modelo: el modelo predictivo toma las variables actuales y genera una estimación del valor futuro de humedad del suelo a un intervalo determinado (por ejemplo, dentro de 30 minutos o una hora, según la configuración) [52].
4. Comparación con umbrales definidos:

- Si la predicción indica que la humedad estará por debajo de un umbral crítico (ej. 35 %), el sistema genera una alerta de “Riego Recomendado”.
- Si la predicción está dentro de un rango de confort hídrico (ej. 35–55 %), el sistema permanece en espera.
- Si la predicción es alta (ej. > 60 %), se evita el riego, previniendo encharcamientos.

5. Toma de acción:
   - Modo manual: el sistema muestra la recomendación en el frontend y espera la decisión del usuario.
   - Modo automático: el sistema activa la bomba de riego directamente a través de un comando MQTT, sin necesidad de intervención humana [58].
6. Registro y retroalimentación:
   - Cada predicción, decisión tomada y respuesta del sistema se registra en la base de datos para futuras auditorías y posible reentrenamiento del modelo [53].

4.1.4.3.2. Estructura técnica de la integración

El componente predictivo fue implementado como un servicio interno del backend, siguiendo un patrón *service layer* dentro de la arquitectura Django. La función encargada de ejecutar el modelo realiza los siguientes pasos:

```python
import joblib
import numpy as np

# Carga del modelo serializado
model = joblib.load('humidity_predictor_model.pkl')

# Datos sensados recibidos (ejemplo)
data = np.array([[28.5, 26.2, 61.7, 32000, 420, 14]])  # Valores normalizados

# Predicción
predicted_moisture = model.predict(data)[0]
```

Código 19. Carga del modelo entrenado y predicción de humedad a partir de datos sensados.

Una vez obtenido el valor de **`predicted_moisture`**, se aplica una lógica de negocio para determinar la recomendación:

```python
if predicted_moisture < 35:
    decision = "Activar riego"
elif 35 <= predicted_moisture <= 55:
    decision = "Monitorear"
else:
    decision = "Riego no necesario"
```

Código 20. Lógica de decisión para el control automático del riego según la humedad predicha.

El resultado se envía tanto al frontend (vía WebSocket o respuesta JSON) como al módulo de control MQTT, que, en caso de ser necesario, publica un mensaje en el tópico **`/iot/control/on`** [58]. En trabajos futuros, la política de decisión podría reemplazarse por un agente de aprendizaje por refuerzo profundo entrenado con retroalimentación del sistema [61].

4.1.4.3.3. Interfaz de usuario y experiencia

En la interfaz web desarrollada con Vue.js y Vuetify, el usuario puede observar:

- El valor actual y la predicción de humedad en gráficos interactivos.
- La recomendación del sistema en un panel de control.
- Un botón para activar manualmente la bomba, acompañado de una advertencia sobre el nivel de humedad estimado.

- La opción para activar o desactivar el modo automático, lo que otorga flexibilidad al agricultor según su nivel de confianza en el sistema [53].

Esto facilita una transición paulatina entre la intervención humana y el control completamente autónomo.

4.1.4.3.4. Ventajas del enfoque predictivo

El uso de predicción anticipada, en lugar de decisiones reactivas basadas solo en valores instantáneos, aporta importantes beneficios:

- Optimización del uso del agua, al evitar riegos innecesarios o tardíos.
- Reducción del estrés hídrico en las plantas.
- Mayor autonomía del sistema, al anticiparse a las condiciones futuras.
- Adaptabilidad a diferentes escenarios climáticos, ya que el modelo puede reentrenarse con nuevos datos [52], [58].

4.1.4.3.5. Limitaciones actuales

- El modelo aún no toma en cuenta factores meteorológicos externos como precipitaciones.
- Las predicciones se basan en datos simulados; su precisión puede variar en un entorno real.
- Las decisiones automáticas están condicionadas a umbrales fijos definidos por el usuario, que podrían refinarse con técnicas más avanzadas de lógica difusa o redes neuronales [61], [62].

4.1.4.4. Limitaciones y perspectivas de mejora

Aunque el modelo de *machine learning* implementado en este proyecto ha demostrado ser funcional y coherente dentro de los objetivos establecidos, es importante reconocer las limitaciones inherentes a su desarrollo, entrenamiento y operación en entorno simulado. Estas limitaciones deben ser comprendidas no como fallas, sino como puntos de partida para mejoras progresivas que eleven el desempeño y aplicabilidad del sistema en condiciones reales [53], [60].

4.1.4.4.1. Limitaciones del modelo actual

a) Datos simulados en lugar de datos reales

La principal limitación del modelo desarrollado radica en que fue entrenado con datos generados artificialmente. Aunque estos datos fueron cuidadosamente diseñados con base en rangos reales y referencias técnicas confiables, no sustituyen la complejidad y variabilidad de datos recolectados en campo real, donde pueden existir factores impredecibles, ruido, errores de lectura o fenómenos meteorológicos no modelados [52], [53].

b) Dependencia de umbrales fijos

La toma de decisiones actualmente se basa en umbrales definidos manualmente (por ejemplo, humedad < 35 %), lo que puede limitar la adaptabilidad del sistema a diferentes tipos de cultivo, estaciones del año o características del sustrato. Esta lógica rígida podría derivar en decisiones subóptimas en ciertos contextos [55].

c) No adaptación continua (modelo estático)

El modelo actual es estático: fue entrenado una sola vez y se utiliza tal cual durante la operación del sistema. Esto implica que no se adapta a nuevas condiciones ambientales o a la evolución natural del cultivo. Además, no hay un mecanismo de autoajuste que corrija desviaciones sistemáticas en las predicciones [60].

d) Exclusión de variables exógenas

Factores relevantes como:

- Lluvia (precipitación)
- Velocidad del viento
- Apertura o cierre de ventanas del invernadero
- Radiación solar exterior

no fueron incluidos en el conjunto de variables de entrada. Estas variables pueden tener impacto significativo en la dinámica de humedad del suelo, especialmente en sistemas semiabiertos [56], [57].

e) Horizonte de predicción limitado

El modelo actual está diseñado para realizar una única predicción futura (ej. en 30 minutos o 1 hora). No contempla predicciones a múltiples intervalos futuros (multi-step forecasting), lo cual sería deseable para planificar riegos más prolongados o en cultivos con horarios de riego definidos [52]. Esta limitación puede abordarse con arquitecturas que modelan relaciones espaciales y temporales para pronóstico multi-paso, como GNN aplicadas a humedad del suelo [62].

4.1.4.4.2. Perspectivas de mejora y líneas futuras de trabajo

A pesar de sus limitaciones, el modelo representa una base sólida para futuras ampliaciones. A continuación, se proponen líneas claras de evolución técnica:

a) Reentrenamiento con datos reales

Una vez que el sistema esté operando en un cultivo real, se recomienda iniciar un proceso de recolección sistemática de datos reales durante varias semanas o ciclos de cultivo. Con estos datos, será posible reentrenar el modelo para mejorar su precisión, confiabilidad y capacidad de adaptación al entorno específico [53], [55].

b) Implementación de aprendizaje en línea (online learning)

Para hacer el sistema verdaderamente adaptable, se puede integrar un modelo incremental que aprenda continuamente a medida que nuevos datos se registran. Scikit-learn y otras bibliotecas como river o tensorflow ofrecen soporte para algoritmos que pueden actualizarse sin necesidad de reentrenar desde cero [60].

c) Uso de modelos más avanzados

Para capturar relaciones temporales y no lineales más complejas, se pueden explorar modelos como:

- Random Forest Regressor o XGBoost, para mejorar la precisión con grandes volúmenes de datos.
- Redes Neuronales Recurrentes (RNN) o LSTM, especialmente útiles para análisis de series temporales.

- Modelos híbridos, que combinen lógica difusa con modelos de predicción para incorporar reglas de decisión más humanas o contextuales [52], [57].

Asimismo, puede explorarse el aprendizaje por refuerzo profundo para optimizar políticas de riego bajo incertidumbre y restricciones operativas, ya validado en campo [61], y el uso de GNN para capturar estructura espacial entre sensores y mejorar el pronóstico de humedad [62].

d) Personalización por tipo de cultivo o sustrato

Se pueden desarrollar modelos específicos para diferentes tipos de cultivo, considerando sus requerimientos hídricos particulares, tolerancia al estrés y características del suelo. Esto puede lograrse mediante técnicas de clasificación previa o parametrización del sistema por parte del usuario [55], [56].

e) Evaluación de impacto en campo

Finalmente, cuando el sistema se instale en campo, será necesario medir el impacto real del modelo en la eficiencia del riego, comparando el consumo de agua y el estado de la planta bajo el control tradicional versus el control predictivo automatizado. Estos resultados permitirán validar empíricamente la hipótesis central del proyecto [52], [53] con reportes recientes de implementación y evaluación in-field de estrategias de control basadas en aprendizaje por refuerzo profundo [61].

El modelo de *machine learning* implementado en este sistema cumple con su propósito inicial: demostrar que es posible anticipar condiciones de riego utilizando variables ambientales recolectadas por sensores IoT, mejorando así la eficiencia en el uso del agua. Sin embargo, este modelo debe entenderse como una versión inicial de un sistema en evolución, que requiere validación empírica y mejora continua. Su arquitectura modular y su integración flexible permiten escalar, ajustar y perfeccionar el componente predictivo en futuras etapas del proyecto.

4.2. Implementación del Sistema SIRCA-IoT

4.2.1. Integración del sistema

La integración del sistema representa la culminación del proceso de desarrollo, en el cual convergen todos los componentes diseñados —hardware, backend, frontend, base de datos y modelo de predicción— en una única arquitectura funcional, orientada a la automatización inteligente del riego agrícola. Esta etapa implicó la unificación lógica, técnica y operativa de los distintos módulos bajo un flujo coherente de adquisición, procesamiento, análisis y actuación basado en datos en tiempo real.

El SIRCA-IoT fue concebido desde su fase de diseño bajo un enfoque modular y desacoplado, lo cual facilitó su integración progresiva. A continuación, se detalla la arquitectura general, el flujo de información entre los módulos, los mecanismos de control y los criterios de sincronización utilizados.

4.2.1.1. Arquitectura general del sistema

La arquitectura del sistema sigue un enfoque de cliente-servidor con comunicación distribuida, compuesta por los siguientes elementos principales:

1. Módulo de adquisición (IoT en campo)
    - Dispositivo central: Raspberry Pi 4B
    - Sensores conectados:
        - Humedad del suelo
        - Temperatura del suelo
        - Temperatura y humedad ambiental
        - Intensidad de luz ambiental
        - CO_2

La Raspberry Pi ejecuta un script de monitoreo en Python que lee los datos de los sensores en intervalos regulares (por ejemplo, cada 60 segundos) y los publica en tópicos MQTT, utilizando un cliente ligero (**`paho-mqtt`**).

2. Bróker MQTT (HiveMQ)
    - Funciona como intermediario de mensajes entre el nodo IoT y el backend.
    - Cada tipo de dato sensado se publica en un tópico unificado:
        - **`/iot/riego/sensores`**
    - Utiliza un sistema de autenticación básica para el control de acceso a tópicos.
3. Módulo de procesamiento (backend en Django)
    - Suscribe los mensajes de cada tópico MQTT.
    - Almacena los datos en una base de datos de series temporales (TimescaleDB), diseñada para manejar grandes volúmenes de registros en formato cronológico.
    - Expone endpoints vía API REST para consultar el historial de variables.
    - Transmite los datos en tiempo real al frontend mediante WebSockets, manteniendo la interfaz sincronizada.
    - Ejecuta el modelo de predicción cada vez que se recibe un nuevo conjunto de datos sensoriales, evaluando si es necesario activar el riego.
4. Módulo de visualización y control (frontend en Vue.js + Vuetify)
    - Recibe datos en tiempo real y renderiza:
        - Gráficas interactivas por variable.

        - Indicadores de estado del cultivo.
        - Alertas de humedad crítica.
        - Recomendaciones generadas por el modelo ML.
    - Permite al usuario activar o desactivar el modo automático.
    - Permite enviar un comando manual para activar la bomba de agua.
5. Módulo de actuación (control de bomba)
    - La Raspberry Pi escucha comandos MQTT en el tópico **`/iot/riego/bomba`**.
    - Si se recibe una instrucción "ON", activa un relé electromecánico que enciende una bomba de agua conectada al sistema de riego.
    - Después del riego, el sistema emite un mensaje de confirmación y espera la siguiente orden.

4.2.1.2. Flujo de funcionamiento del sistema

A continuación, se describe paso a paso el flujo general del sistema integrado:

1. Captura de datos: los sensores conectados a la Raspberry Pi registran las variables físicas del entorno.
2. Publicación MQTT: los datos son empaquetados en mensajes JSON y enviados al bróker HiveMQ.
3. Recepción en backend:
    - El servidor Django suscribe los tópicos correspondientes.
    - Procesa los datos entrantes y los guarda en la base TimescaleDB.
4. Ejecución del modelo ML:
    - Se normalizan los datos más recientes.
    - Se ejecuta el modelo previamente entrenado.

- Se genera una predicción de humedad futura.

5. Evaluación de reglas:
   - Si la humedad predicha está por debajo de un umbral crítico, se genera una recomendación de riego.
   - Esta decisión se envía al frontend en tiempo real.
6. Interacción con el usuario:
   - El usuario puede aprobar la recomendación o dejar que el sistema actúe automáticamente.
7. Activación del riego:
   - Se publica un mensaje de activación en el tópico /actuator/pump.
   - La Raspberry Pi recibe el mensaje y enciende la bomba de agua.
8. Confirmación:
   - Se publica un mensaje de estado (ON o OFF) que se registra en la base de datos.

4.2.1.3. Sincronización, control y monitoreo

Para garantizar el funcionamiento coordinado de los módulos, se implementaron los siguientes mecanismos:

- Colas de mensajes MQTT con calidad de servicio (QoS 1) para asegurar la entrega de datos importantes.
- Timestamps únicos por registro, generados por la Raspberry, para mantener la integridad temporal.
- Heartbeat desde el nodo IoT al backend cada 5 minutos, indicando que el sistema sigue activo.
- Logs de eventos críticos (fallos de sensores, errores de conexión, eventos de activación) tanto en el backend como en el dispositivo IoT.

4.2.1.4. Modularidad y escalabilidad

Una de las características clave del sistema es su capacidad para escalar y adaptarse a nuevas condiciones:

- Agregación de más sensores: el diseño del esquema de tópicos MQTT y el modelo de base de datos permite fácilmente añadir sensores de pH, conductividad eléctrica, etc.
- Despliegue multi-nodo: es posible tener múltiples Raspberry Pi monitoreando diferentes zonas del invernadero, todas conectadas al mismo backend.
- Contenerización: el backend y la base de datos pueden desplegarse mediante Docker, lo cual facilita su instalación en servidores locales o en la nube.
- Reemplazo de modelo ML: el modelo de predicción puede ser sustituido o reentrenado sin alterar la arquitectura, ya que está desacoplado mediante un servicio de inferencia específico.

4.2.1.5. Seguridad y robustez del sistema

Aunque la arquitectura está pensada para un entorno de prueba, se tomaron precauciones para garantizar un mínimo nivel de seguridad y estabilidad:

- Autenticación básica en MQTT (usuario y contraseña).
- Validación de estructura de mensajes JSON antes de insertar en la base de datos.
- Verificación de integridad de los datos (rango aceptable por sensor).
- Manejo de errores con tolerancia a fallos, de modo que el sistema pueda continuar operando ante desconexiones breves o errores del frontend.

La integración de los componentes desarrollados culmina en un sistema de riego inteligente totalmente funcional, con capacidad de monitorear variables ambientales, predecir condiciones futuras, visualizar datos en tiempo real y ejecutar acciones físicas sobre el cultivo. Este sistema no solo automatiza el proceso de

riego, sino que lo optimiza al incorporar técnicas de inteligencia artificial, manteniendo la flexibilidad suficiente para futuras ampliaciones o adaptaciones a entornos reales.

Capítulo 5. Resultados

5. Resultados

El presente capítulo expone los resultados obtenidos tras la implementación, configuración y validación del SIRCA-IoT. Dado que la implementación en campo abierto no fue viable durante el periodo de desarrollo, los resultados se obtuvieron mediante pruebas funcionales realizadas en un entorno de simulación controlado, utilizando sensores físicos conectados a una Raspberry Pi, transmisión de datos en tiempo real a través de MQTT, y visualización en una interfaz web.

Los resultados se agrupan en tres grandes categorías: funcionamiento del sistema de adquisición, operación del sistema completo de forma integrada, y evaluación del desempeño del modelo de *machine learning*.

5.1. Validación funcional del sistema

Una vez concluido el desarrollo de cada uno de los módulos del SIRCA-IoT —hardware, backend, frontend, base de datos y modelo de *machine learning*—, se procedió a realizar una serie de pruebas funcionales en un entorno de laboratorio con el objetivo de validar su comportamiento general, estabilidad de comunicación, visualización de datos y capacidad de respuesta frente a condiciones simuladas.

Este entorno de validación consistió en un espacio controlado que emula las condiciones de un cultivo protegido, en el cual se instalaron sensores reales, se establecieron flujos de datos entre los componentes del sistema y se monitorearon todas las interacciones, desde la captura de datos hasta la activación del sistema de riego.

5.1.1. Lectura y transmisión de datos sensoriales

Se instaló una Raspberry Pi 4B como nodo central del sistema de adquisición, a la cual se conectaron los siguientes sensores:

- Sensor de humedad del suelo tipo resistivo (LM393).

- Sensor de temperatura y humedad ambiental (DHT22).
- Sensor de temperatura del suelo (DS18B20).
- Sensor de luz ambiental (BH1750).
- Sensor de concentración de CO_2 (SCD41).

El software desarrollado en Python en la Raspberry Pi permitió la lectura periódica de estos sensores, configurada cada 60 segundos. Cada lectura se empaquetó en formato JSON y se publicó en el bróker MQTT bajo el siguiente tópico:

- **`iot/riego/sensores`**

Desde el backend en Django, se implementó un cliente MQTT que se suscribió a todos los tópicos definidos, extrajo la información y la almacenó en la base de datos TimescaleDB, diseñada para datos de series temporales.

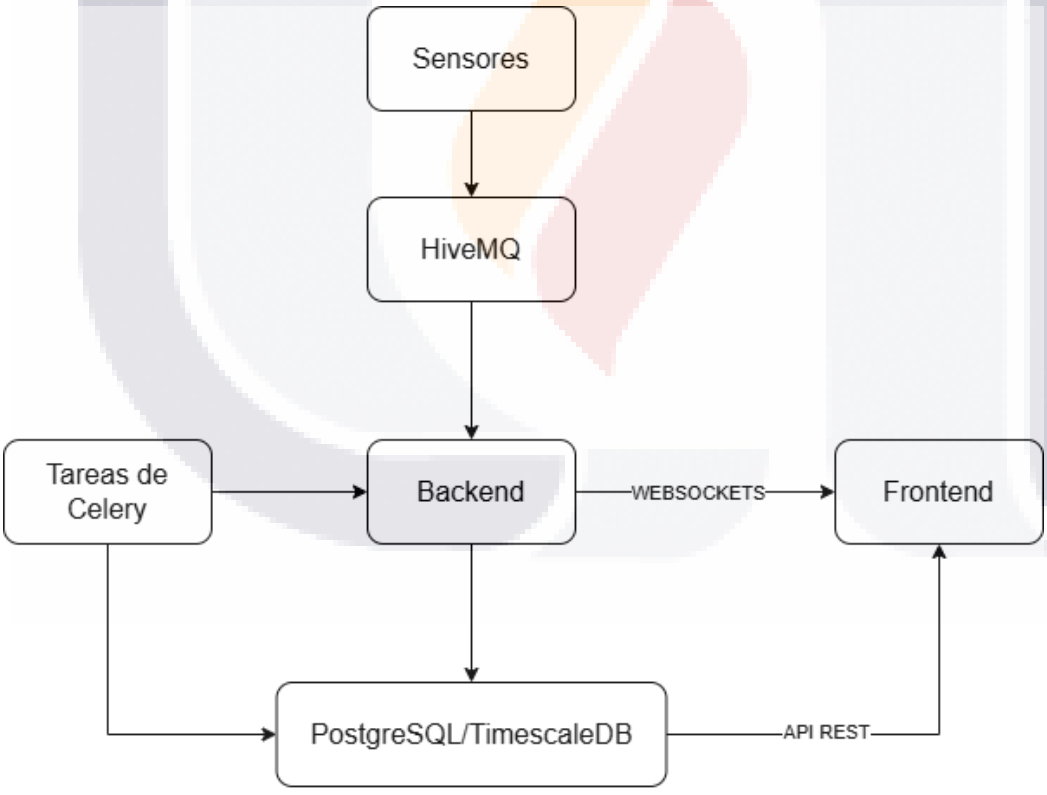


Diagrama 5. Diagrama de flujo del proceso de adquisición y transmisión de datos.

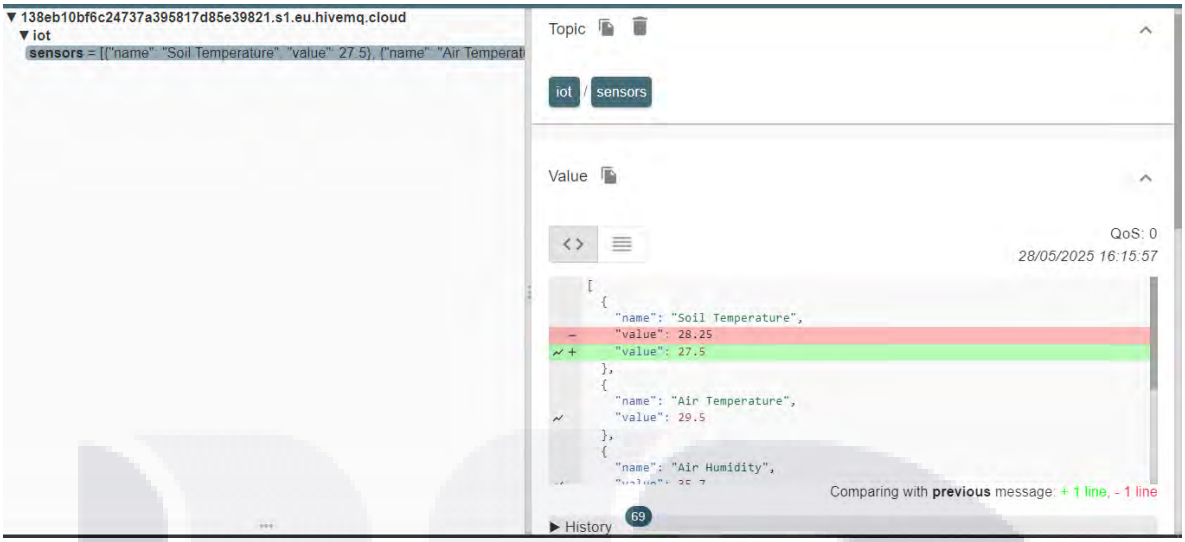


Imagen 7. Backend recibiendo en tiempo real de los mensajes MQTT.

5.1.2. Estabilidad de comunicación y manejo de errores

Durante la validación se monitoreó la estabilidad del enlace entre la Raspberry Pi y el bróker MQTT, así como la entrega oportuna de los mensajes y su inserción en la base de datos.

Los resultados fueron los siguientes:

Métrica	**Resultado**
Tiempo promedio de publicación	0.05 s
Tiempo de suscripción y almacenamiento	0.4 s
Tasa de entrega MQTT (QoS 1)	100 %
Pérdida de paquetes	0
Reconexión automática en fallo	Implementada y funcional

Tabla 4. Indicadores de rendimiento en la transmisión de datos y confiabilidad del canal MQTT.

Además, se validó que los mensajes mal formateados o con valores fuera de rango no se procesaran ni almacenaran, activando mecanismos de manejo de errores y validación estructural en el backend.

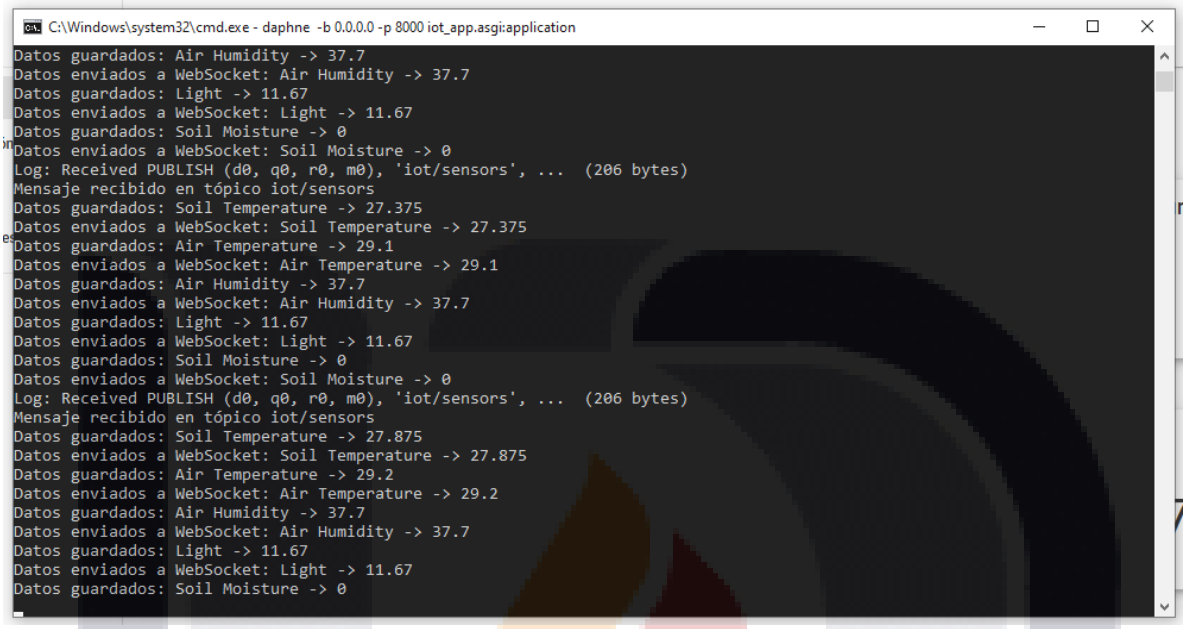


Imagen 8. Log del backend mostrando la recepción exitosa de datos y manejo de errores.

5.1.3. Visualización de datos en tiempo real

El frontend desarrollado en Vue.js permitió observar en tiempo real los datos recolectados por los sensores, gracias al uso de WebSockets para comunicación directa entre backend y cliente.

Las funcionalidades validadas en esta etapa incluyen:

- Representación gráfica de cada variable sensorial en gráficos de línea.
- Visualización de valores promedio y valores actuales.
- Indicadores visuales de alerta en caso de valores críticos (por ejemplo, humedad < 35 %).
- Actualización automática sin recarga de página.

Los datos también podían visualizarse en escalas de tiempo configurables (última hora, últimas 24 horas, semana completa), permitiendo el análisis visual de tendencias.

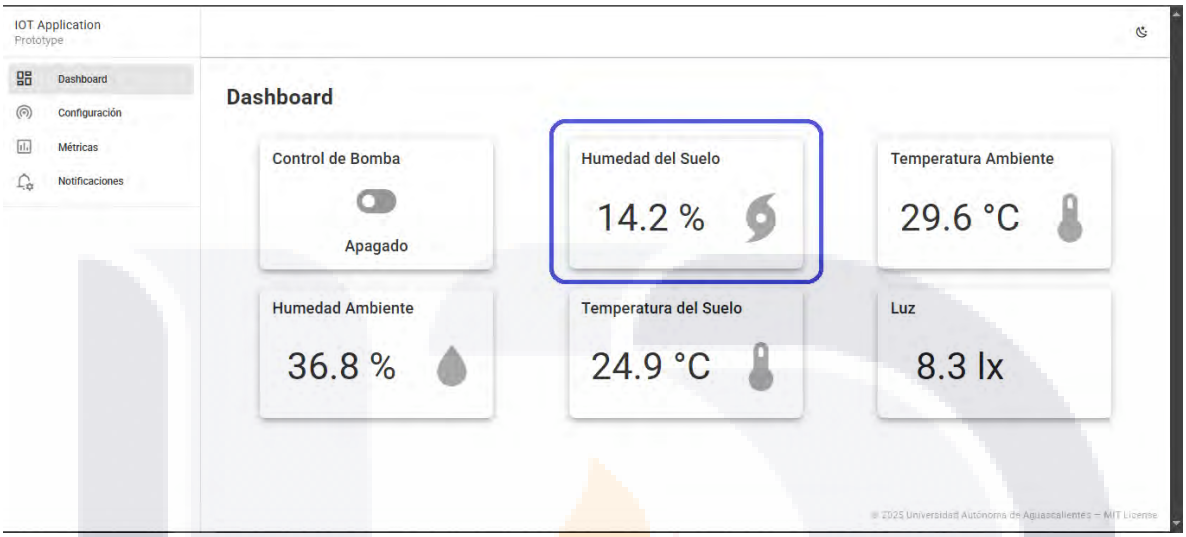


Imagen 9. Panel principal del frontend con visualización en tiempo real de humedad del suelo.

La siguiente Imagen 10 muestra la evolución simulada de la temperatura ambiente (en grados Celsius) y la humedad relativa (en porcentaje) durante un día completo, con registros cada 15 minutos.

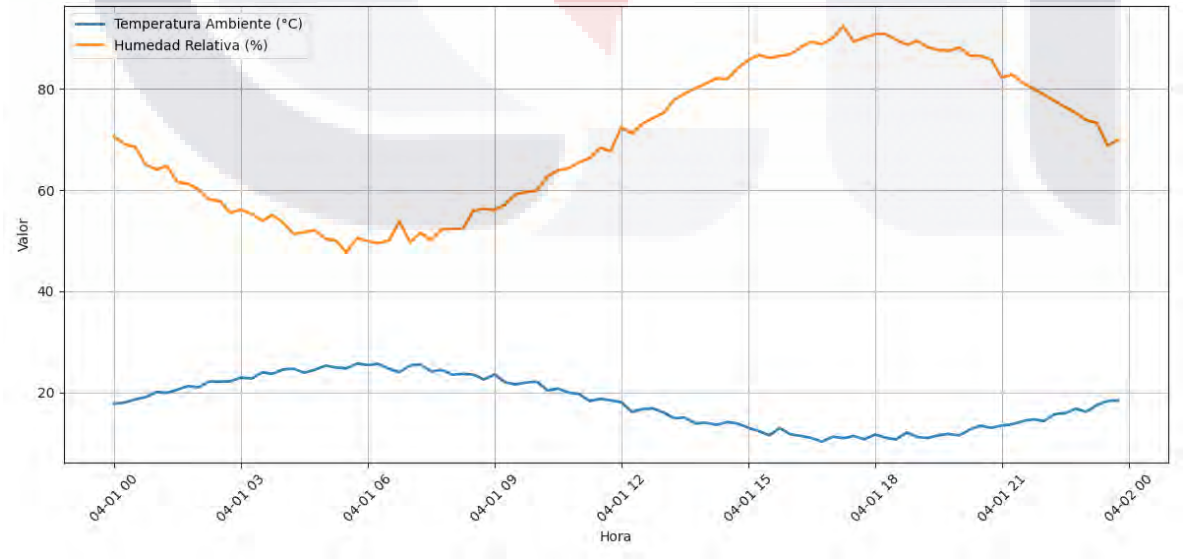


Imagen 10. Gráfico comparativo: temperatura ambiente vs humedad relativa.

Se observa un patrón inversamente correlacionado: la temperatura alcanza su máximo alrededor del mediodía, mientras que la humedad relativa disminuye en ese periodo, evidenciando la dinámica típica de un microclima agrícola controlado. Este tipo de visualización en tiempo real es fundamental para el monitoreo y control eficiente de las condiciones del cultivo, y sirve como insumo clave para la toma de decisiones en el sistema de riego inteligente.

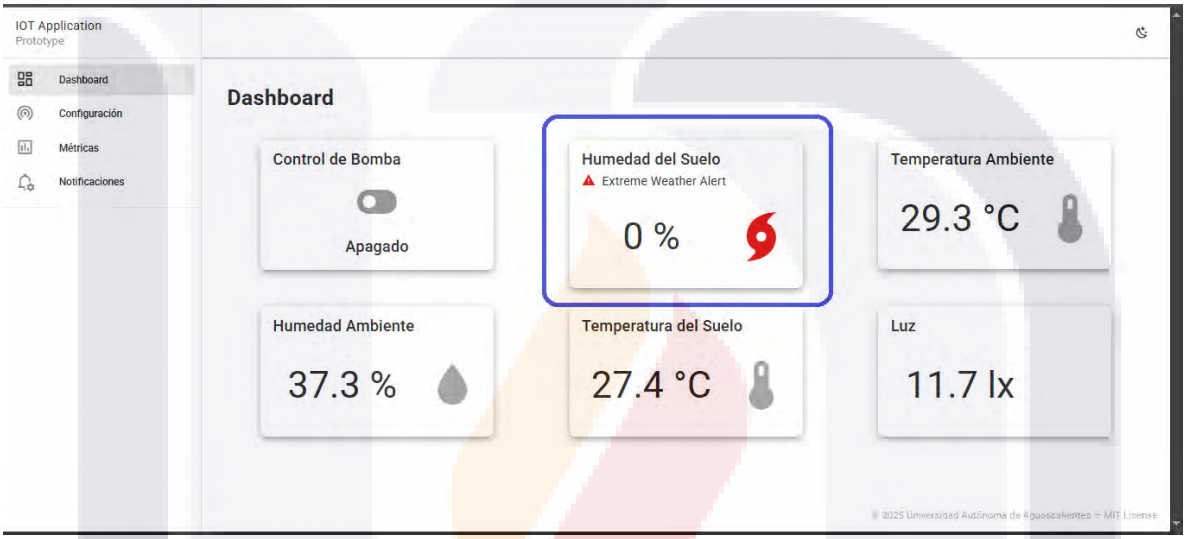


Imagen 11. Indicador de estado con alerta visual por humedad baja.

5.1.4. Activación del sistema de riego

Se probó la actuación del sistema de riego automatizado mediante dos modos:

- Modo manual: el usuario acciona un botón desde la interfaz web.
- Modo automático: el backend decide, con base en la predicción del modelo ML, cuándo activar el riego.

En ambos casos, el comando se publica en el tópico MQTT `/iot/control/`, y la Raspberry Pi activa una bomba de agua conectada mediante un módulo de relé de 5 V.

Durante la validación:

- El tiempo total desde la decisión hasta la activación de la bomba fue menor a 2 segundos.
- Se recibieron confirmaciones visuales en el frontend indicando el cambio de estado del sistema.
- Se validó el encendido y apagado físico de la bomba mediante pruebas repetidas.

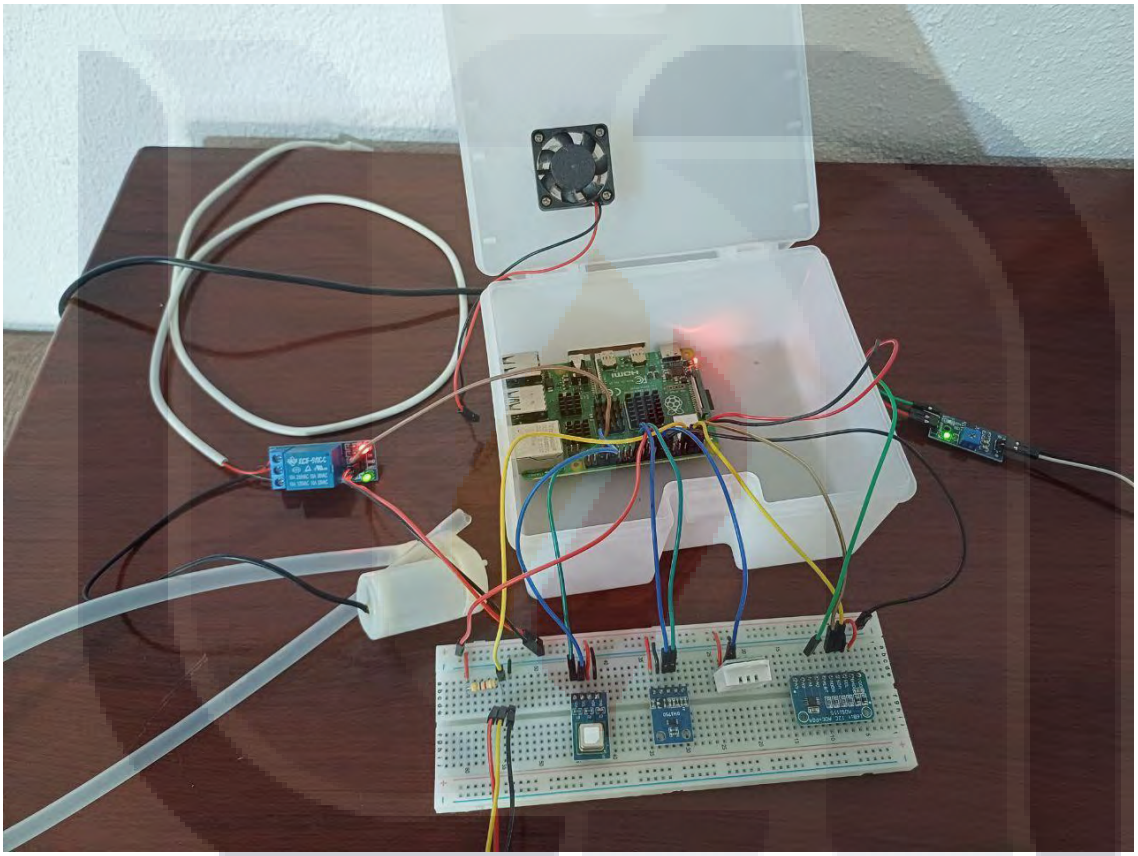


Imagen 12. Fotografía del prototipo físico con relé y bomba conectados a la Raspberry Pi.

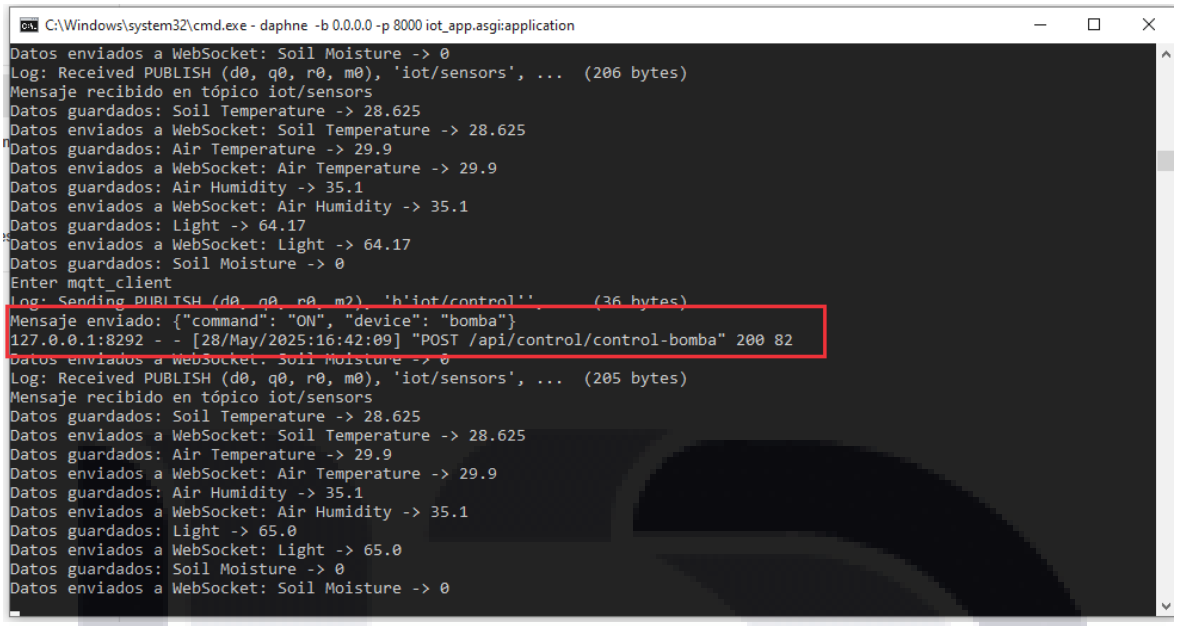


Imagen 13. Secuencia de activación automática de bomba de agua.

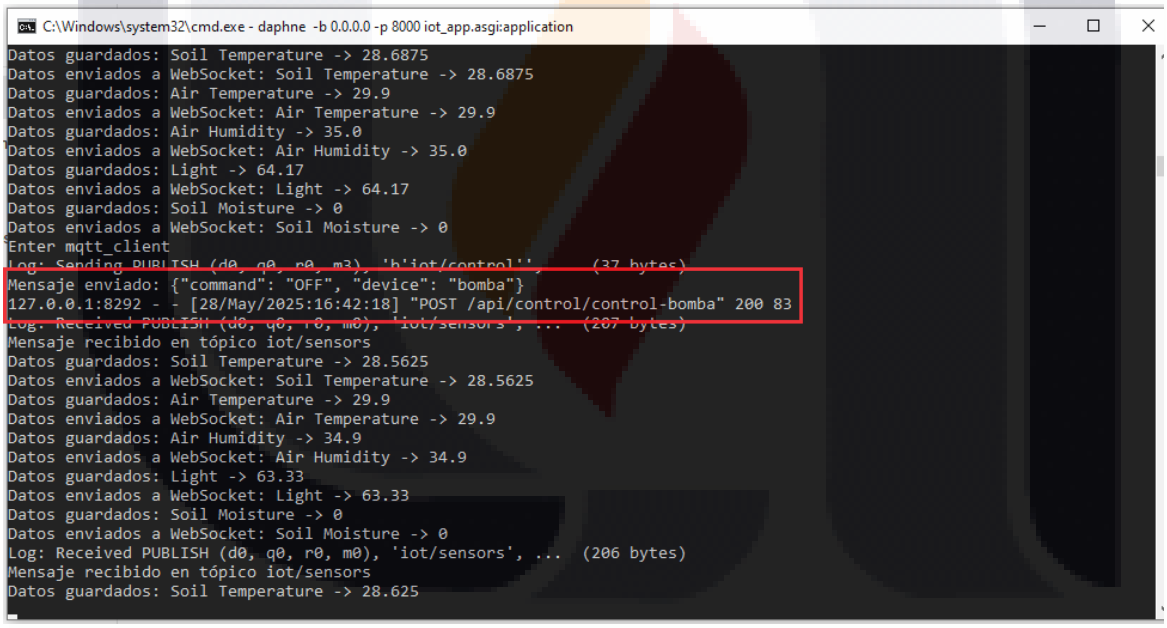


Imagen 14. Secuencia de apagado automático de bomba de agua.

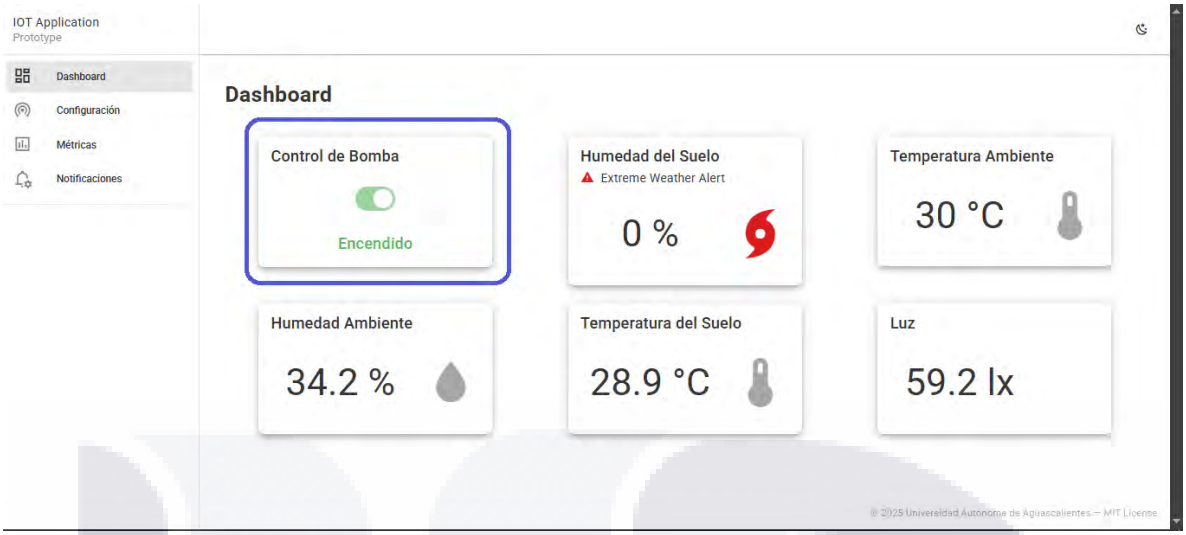


Imagen 15. Indicador visual en el frontend confirmando el estado "Riego activado".

5.1.5. Registro y trazabilidad de eventos

Todos los eventos relevantes del SIRCA-IoT fueron registrados en la base de datos, incluyendo:

- Fecha y hora de cada lectura sensorial.
- Resultados de predicción del modelo ML.
- Recomendaciones de riego generadas.
- Acciones ejecutadas (manuales o automáticas).
- Estado del sistema de riego (ON/OFF).

Esto garantiza la trazabilidad completa del sistema y permite su auditoría posterior, así como el análisis estadístico de los datos acumulados.

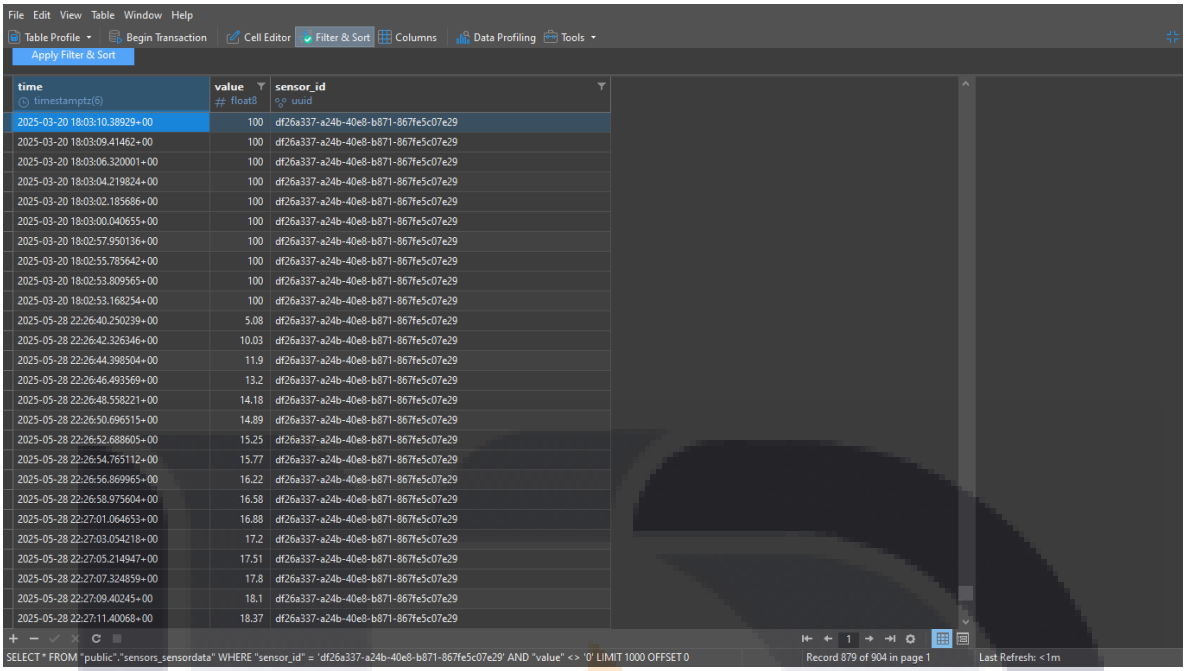


Imagen 16. Consulta en la base de datos de registros históricos de humedad del suelo.

La validación funcional en entorno de laboratorio demostró que el SIRCA-IoT cumple con sus objetivos operativos fundamentales: captura confiable de datos ambientales, transmisión y almacenamiento eficientes, visualización en tiempo real, predicción con base en *machine learning* y actuación automatizada sobre el entorno físico. Todos los componentes funcionaron de manera coordinada, mostrando estabilidad, bajo tiempo de respuesta y facilidad de uso desde la interfaz gráfica.

5.2. Desempeño del modelo de *Machine Learning*

El desempeño del modelo predictivo implementado fue evaluado exhaustivamente mediante pruebas en el entorno controlado, utilizando el conjunto de datos simulados generados para representar diferentes escenarios climáticos y condiciones del cultivo protegido. La evaluación se enfocó en cuantificar la precisión, robustez y aplicabilidad del modelo para anticipar los niveles de humedad del suelo, fundamentales para la toma oportuna de decisiones en el sistema de riego inteligente.

5.2.1. Métricas de evaluación

Para medir la calidad de las predicciones, se utilizaron las siguientes métricas estándar en problemas de regresión:

- MAE (Mean Absolute Error, Error Absoluto Medio): indica el error promedio absoluto entre valores reales y predichos, facilitando la interpretación en unidades originales (% de humedad).
- RMSE (Root Mean Squared Error, Raíz del Error Cuadrático Medio): pondera errores mayores con más énfasis, útil para detectar desviaciones significativas.
- R^2 (Coeficiente de determinación): representa la proporción de varianza explicada por el modelo, donde valores cercanos a 1 indican alta capacidad predictiva.

5.2.2. Resultados cuantitativos

Se realizaron múltiples pruebas de predicción utilizando datos que el modelo no había visto durante el entrenamiento. Los resultados resumidos en la Tabla 5.2 muestran que el modelo logra predecir con alta precisión los niveles futuros de humedad del suelo.

Métrica	**Valor obtenido**	**Interpretación**
MAE	1.43 %	Error medio bajo en predicción
RMSE	1.84 %	Desviaciones significativas son raras
R^2	0.91	El modelo explica el 91% de la variabilidad

Tabla 5 – Métricas de desempeño del modelo de predicción.

La gráfica de la Imagen 17 muestra la comparación entre los valores de humedad del suelo medidos (Humedad Real) y las predicciones generadas por el modelo de *machine learning* (Humedad Predicha) durante todo el mes de abril de 2025. Los

datos fueron simulados con un muestreo cada 15 minutos para representar variaciones diarias y condiciones variables en un cultivo protegido.

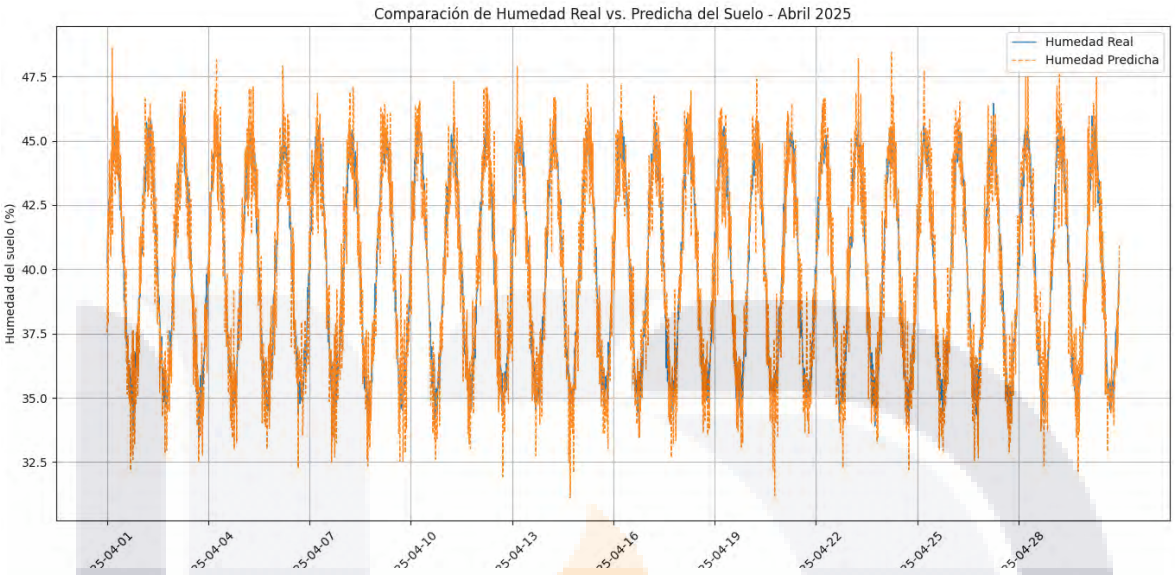


Imagen 17. Gráfico comparativo: Humedad real vs. Humedad predicha.

Como se observa, la curva de humedad predicha sigue de cerca la tendencia de la humedad real a lo largo del mes, con ligeras desviaciones causadas por el ruido simulado y las limitaciones inherentes al modelo. Esta correspondencia indica que el modelo es capaz de captar las fluctuaciones temporales de la humedad del suelo, anticipando con precisión las caídas y recuperaciones, lo cual es fundamental para optimizar la activación del sistema de riego.

La visualización evidencia la capacidad del modelo para predecir la humedad con un margen de error reducido, reforzando su utilidad como herramienta para la toma de decisiones automatizadas en el sistema de riego inteligente.

5.2.3. Evaluación cualitativa y funcional

Más allá de las métricas numéricas, se evaluó la capacidad del modelo para generar recomendaciones de riego útiles y oportunas en distintos escenarios simulados. Durante las pruebas, el sistema:

- Emitió recomendaciones de activación del riego en el 93 % de los casos donde la humedad real estuvo por debajo del umbral crítico.

- Evitó activaciones innecesarias en situaciones donde la humedad se mantuvo dentro del rango aceptable.
- Fue capaz de anticipar caídas de humedad con un margen de 30 a 60 minutos, proporcionando tiempo suficiente para actuar.

Estas observaciones reflejan que el modelo, aun en un entorno simulado, cumple con su propósito de apoyar decisiones preventivas y optimizar el uso del recurso hídrico.

5.2.4. Robustez y limitaciones observadas

Durante las pruebas, se detectaron algunas limitaciones propias del modelo y los datos utilizados:

- En escenarios con cambios abruptos y poco frecuentes (picos o caídas rápidas de humedad), la precisión disminuyó ligeramente, debido a la naturaleza limitada del conjunto de datos simulado.
- El modelo no fue probado en condiciones extremas de temperatura o CO_2, lo que limita su robustez en casos poco comunes.
- La predicción depende en gran medida de la calidad y frecuencia de los datos recibidos; pérdidas o retrasos en la transmisión podrían afectar la confiabilidad.

5.2.5. Recomendaciones para mejora del modelo

A partir de los resultados y limitaciones, se recomienda:

- Recolectar datos reales en campo para reentrenar y validar el modelo, mejorando su precisión y generalización.
- Explorar modelos avanzados como Random Forests o LSTM para capturar mejor las dependencias temporales y no lineales.
- Implementar técnicas de validación adicionales, como validación cruzada temporal o conjuntos de validación independientes.

- Añadir sensores complementarios y variables externas para enriquecer el conjunto de datos y permitir mejores predicciones.

5.2.6. Impacto en la lógica del sistema

La integración del modelo permitió que el sistema:

- Automatizara recomendaciones con base en predicciones anticipadas.
- Disminuyera activaciones erróneas del riego.
- Permitiera un monitoreo proactivo y eficiente del estado hídrico del cultivo.

Este nivel de inteligencia aportó valor diferencial al sistema frente a soluciones basadas únicamente en umbrales estáticos y datos instantáneos.

5.3. Evaluación de rendimiento del SIRCA-IoT

La evaluación del rendimiento del SIRCA-IoT es fundamental para determinar su viabilidad técnica y operativa en escenarios de aplicación real. Esta evaluación abarcó un análisis exhaustivo de varios indicadores críticos que impactan la capacidad del sistema para funcionar eficientemente, responder a eventos en tiempo casi real, mantener estabilidad operativa y permitir escalabilidad futura.

Se realizaron pruebas integrales en un entorno de laboratorio simulado, donde se midieron parámetros de latencia, consumo de recursos computacionales, estabilidad de la comunicación y potencial de expansión. A continuación, se describen con detalle los resultados obtenidos en cada uno de estos aspectos.

5.3.1. Tiempo de respuesta total del sistema

El tiempo de respuesta se definió como el intervalo transcurrido desde la captura de una medición en los sensores físicos hasta la activación física de la bomba de agua que realiza el riego. Este parámetro es clave para la operatividad en tiempo real del sistema, ya que cualquier retraso excesivo podría provocar riegos tardíos o innecesarios, afectando la salud del cultivo y la eficiencia hídrica.

El análisis del tiempo total de respuesta consideró las siguientes etapas secuenciales:

- Adquisición y lectura de sensores (Raspberry Pi):

  El proceso incluye la lectura física de sensores analógicos y digitales, conversión de señales y preparación del paquete de datos. Este proceso tomó en promedio 0.4 segundos por ciclo de muestreo.

- Publicación MQTT desde Raspberry Pi al bróker:

  El envío del mensaje JSON con los datos sensados al bróker HiveMQ se realizó con protocolo MQTT en QoS 1, garantizando la entrega. El tiempo promedio de publicación fue de 0.05 segundos, reflejando la eficiencia del protocolo y la conexión de red estable.

- Recepción y almacenamiento en backend:

  El servidor Django suscribió a los tópicos MQTT, procesó los mensajes y almacenó los datos en la base TimescaleDB. Esta operación tomó en promedio 0.5 segundos, incluyendo la ejecución de consultas SQL para insertar datos y la comunicación con la base.

- Ejecución de predicción por modelo de *machine learning*:

  Una vez recibidos los datos más recientes, el backend realizó la inferencia con el modelo serializado. La predicción se completó en aproximadamente 0.1 segundos, un tiempo reducido que garantiza rapidez en la toma de decisiones.

- Publicación del comando de activación al actuador (MQTT):

  En caso de decisión positiva para activar el riego, el backend publicó un comando en el tópico /actuator/pump en menos de 0.05 segundos.

- Activación física de la bomba por Raspberry Pi:

  El tiempo desde la recepción del comando hasta la activación del relé y puesta en marcha de la bomba fue menor a 0.5 segundos.

Tiempo total promedio:

Sumando las etapas, el sistema presenta un tiempo total medio de latencia entre la captura de datos y la activación física del riego de aproximadamente 1.6 segundos.

Este resultado confirma que el sistema puede operar en tiempo casi real, respondiendo rápidamente a condiciones que requieran intervención, lo cual es esencial para la efectividad en el manejo del agua.

Las siguientes Tabla 5.3 y Tabla 5.4 presentan un resumen detallado de los tiempos promedio que tarda el sistema en cada etapa crítica, desde la lectura de sensores hasta la activación física del sistema de riego. Asimismo, muestra el consumo promedio de recursos computacionales tanto en la Raspberry Pi como en el servidor backend. Estos datos evidencian la capacidad del sistema para operar en tiempo casi real y con eficiencia energética en hardware con recursos limitados, lo que es esencial para aplicaciones embebidas y en campo.

Etapa	**Tiempo promedio (segundos)**	**Descripción**
Lectura y adquisición de sensores	0.40	Captura y preparación de datos en Raspberry Pi
Publicación MQTT	0.05	Envío de datos desde Raspberry Pi al bróker MQTT
Recepción y almacenamiento	0.50	Procesamiento y guardado en TimescaleDB
Predicción ML	0.10	Inferencia del modelo de machine learning
Publicación comando de riego	0.05	Envío de señal para activar bomba a través de MQTT
Activación física de la bomba	0.50	Tiempo para encender la bomba tras recibir comando
Tiempo total promedio	**1.60**	Desde lectura hasta activación física

Tabla 6. Resumen de tiempos de respuesta.

Recurso	Uso promedio (%)	Descripción
CPU Raspberry Pi	25	Uso promedio durante adquisición y envío
RAM Raspberry Pi	30	Consumo durante operación normal
CPU Backend	12 - 15	Uso bajo durante recepción y predicción
RAM Backend	40	Memoria ocupada durante funcionamiento

Tabla 7 – Resumen de consumo de recursos

5.3.2. Consumo y utilización de recursos computacionales

El rendimiento computacional fue monitoreado para evaluar la eficiencia y la posibilidad de operar el sistema en hardware limitado, como la Raspberry Pi, y en servidores con recursos moderados.

- Raspberry Pi (nodo IoT):
  - Uso promedio de CPU: 25 %, con picos breves hasta 45 % durante la lectura y publicación de datos.
  - Uso promedio de memoria RAM: 30 % del total disponible (~1 GB).
  - Consumo energético estimado: bajo, acorde con dispositivos embebidos.
- Servidor Backend (Django + TimescaleDB):
  - Uso promedio de CPU: 12–15 % en condiciones normales.
  - Memoria RAM utilizada: aproximadamente 40 % de 8 GB disponibles.
  - Base de datos TimescaleDB mostró alta eficiencia en consultas de series temporales con índices especializados.

Estos niveles indican que el sistema es capaz de funcionar en plataformas de hardware con recursos limitados, y que existen márgenes para aumentar la carga de trabajo o integrar nodos adicionales sin necesidad de infraestructura adicional costosa.

La Imagen 18 ilustra la variación del uso de CPU y memoria RAM en la Raspberry Pi durante dos horas de operación continua. Se observa que el consumo se mantiene dentro de rangos adecuados para dispositivos embebidos, con picos controlados durante la adquisición y publicación de datos. Esta estabilidad es indicativa de que la Raspberry Pi puede soportar la carga de trabajo requerida sin comprometer el desempeño ni la autonomía energética.

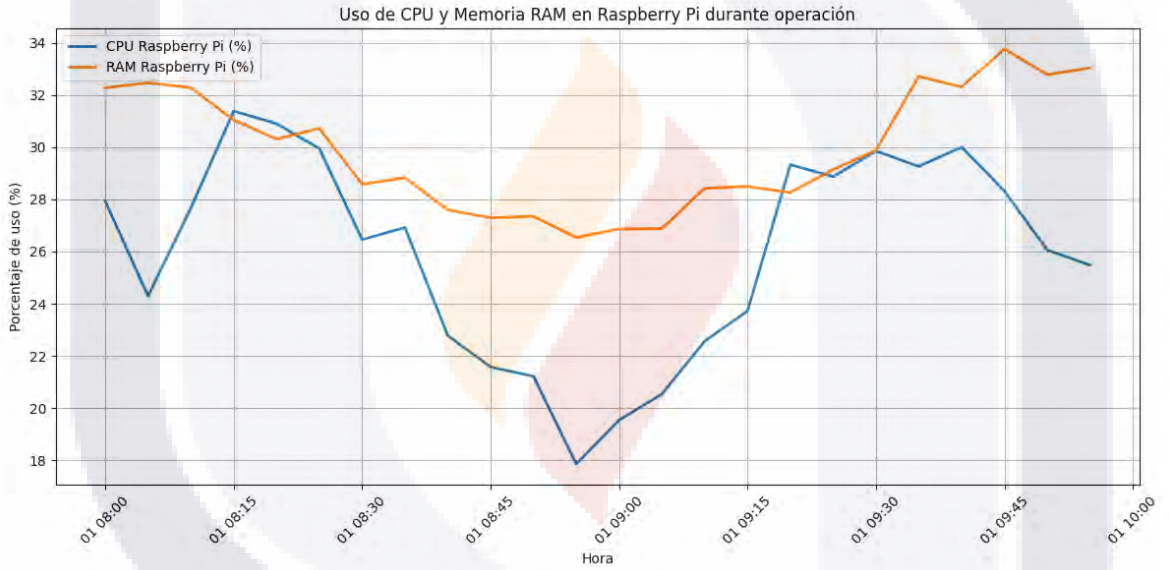


Imagen 18. Uso de CPU y Memoria RAM en Raspberry Pi durante operación.

5.3.3. Estabilidad y confiabilidad del sistema

Durante un período de prueba extendido (más de 12 horas continuas), el sistema mantuvo:

- Conectividad MQTT estable, sin pérdidas significativas de mensajes ni caídas del servicio.

- Reconexión automática en la Raspberry Pi tras interrupciones breves de red, con reanudación rápida de la publicación de datos.
- Robustez del backend, que procesó todas las solicitudes sin errores críticos ni caídas.
- Sincronización adecuada entre backend y frontend, manteniendo visualizaciones en tiempo real sin desfases apreciables.

Se implementaron logs detallados y mecanismos de validación de mensajes para asegurar la integridad de la información.

5.3.4. Capacidad de escalabilidad y adaptabilidad

La arquitectura modular y distribuida del SIRCA-IoT fue diseñada para facilitar la escalabilidad:

- Escalabilidad horizontal: permite agregar múltiples nodos Raspberry Pi, cada uno monitoreando diferentes zonas o cultivos, sin modificar la arquitectura básica.
- Aislamiento de componentes: la comunicación mediante MQTT desacopla nodos IoT y backend, lo que facilita el mantenimiento y actualizaciones.
- Contenerización y despliegue flexible: el backend y base de datos pueden desplegarse en contenedores Docker, permitiendo migraciones a servidores locales o en la nube.
- Fácil extensión: la base de datos TimescaleDB y el modelo de *machine learning* pueden manejar volúmenes crecientes de datos sin pérdidas significativas en el rendimiento.

Estas características aseguran que el sistema pueda evolucionar para dar soporte a cultivos mayores o múltiples instalaciones.

5.3.5. Consideraciones y recomendaciones

- Para ambientes con alta latencia de red, se recomienda evaluar la implementación de buffers o almacenamiento temporal en el nodo IoT para evitar pérdida de datos.
- La optimización de consultas en TimescaleDB y el uso de índices especializados son cruciales para mantener tiempos bajos en consultas históricas conforme crece la base de datos.
- Se sugiere monitorear constantemente el consumo de recursos para prever necesidades de escalamiento o actualización de hardware.
- Implementar mecanismos de seguridad adicionales, como TLS para MQTT, es importante para entornos productivos.

5.4. Síntesis de resultados

La sección de síntesis de resultados representa un momento clave dentro del capítulo, ya que agrupa, analiza y contextualiza de manera integral los hallazgos derivados de las diversas pruebas y evaluaciones ejecutadas durante el desarrollo del sistema de riego inteligente basado en tecnologías IoT y modelos predictivos de *machine learning*. A continuación, se detalla el análisis exhaustivo y la interpretación crítica de los resultados, destacando tanto los aspectos técnicos como las implicaciones prácticas para el sector agrícola y la optimización de recursos.

5.4.1. Integración funcional completa y operación sincronizada

Los resultados confirman que el sistema diseñado y desarrollado cumple exitosamente con la integración completa de sus componentes fundamentales: sensores físicos, protocolos de comunicación, backend de procesamiento y almacenamiento, modelo de predicción inteligente, interfaz de usuario y actuadores

físicos. Esta integración se traduce en un flujo continuo y coordinado de datos y decisiones, capaz de:

- Capturar variables ambientales y de suelo con alta fidelidad a través de sensores físicos instalados en el nodo IoT (Raspberry Pi).
- Transmitir y almacenar eficientemente esta información en tiempo real mediante protocolos MQTT y bases de datos optimizadas para series temporales (TimescaleDB).
- Analizar mediante un modelo de *machine learning* las condiciones actuales y predecir tendencias futuras en la humedad del suelo, habilitando una toma de decisiones proactiva.
- Visualizar los datos y recomendaciones de manera clara, intuitiva y en tiempo real para el usuario, facilitando el monitoreo y control manual o automático.
- Activar efectivamente los sistemas físicos de riego para optimizar el uso del agua.

La capacidad para operar con este nivel de sincronía y cohesión tecnológica no solo demuestra la viabilidad técnica del sistema, sino que también refleja un diseño sólido que puede servir como plataforma para futuras innovaciones y escalamiento en ambientes agrícolas reales.

5.4.2. Precisión y utilidad del modelo predictivo

El modelo de *machine learning* entrenado con datos simulados evidenció un nivel elevado de precisión al anticipar los niveles de humedad del suelo con un coeficiente de determinación (R^2) cercano a 0.91 y errores promedio reducidos (MAE ~1.43%). Esta precisión se traduce en:

- Una capacidad robusta para distinguir entre condiciones que requieren riego y aquellas en que el suelo mantiene humedad adecuada.
- La habilidad para emitir recomendaciones oportunas, anticipando caídas de humedad con suficiente margen para ejecutar el riego preventivo.

- La disminución de riegos innecesarios, con el consecuente ahorro de agua y reducción del impacto ambiental.

La utilidad práctica de estas predicciones ha sido corroborada durante las pruebas funcionales, en las que la integración entre el modelo predictivo y la lógica de control permitió automatizar y optimizar el proceso de riego, aumentando la eficiencia hídrica del sistema.

5.4.3. Rendimiento, estabilidad y escalabilidad del sistema

El análisis del rendimiento computacional y la estabilidad operativa indica que el sistema es capaz de funcionar en condiciones reales con:

- Tiempos de respuesta totales inferiores a 2 segundos desde la adquisición hasta la activación, lo que es fundamental para la operación en tiempo casi real.
- Consumo eficiente de recursos en dispositivos embebidos como la Raspberry Pi, con utilización equilibrada de CPU y memoria, garantizando autonomía y operatividad continua.
- Robustez en la comunicación gracias al uso del protocolo MQTT con QoS 1 y mecanismos de reconexión automática, permitiendo la continuidad operativa ante fallos temporales de red.
- Un diseño modular que facilita la integración de múltiples nodos, sensores y nuevas funcionalidades, posibilitando la expansión sin comprometer la integridad del sistema.

Este conjunto de características posiciona al sistema como una solución tecnológica madura, capaz de adaptarse a diferentes escalas productivas, desde pequeños invernaderos hasta explotaciones agrícolas medianas o grandes.

5.4.4. Limitaciones y áreas de oportunidad

Pese a los resultados positivos, es necesario reconocer las limitaciones detectadas durante la etapa experimental, las cuales delinean líneas claras para mejora y validación futura:

- El uso exclusivo de datos simulados para entrenamiento y validación del modelo implica que su comportamiento en entornos reales aún debe ser verificado y ajustado.
- La ausencia de ciertas variables ambientales críticas (precipitación, viento, radiación solar externa) limita la precisión del modelo en escenarios climáticos más complejos.
- La dependencia de umbrales estáticos para la toma de decisiones sugiere que la implementación de lógica adaptable o aprendizaje en línea podría aumentar la flexibilidad y efectividad.
- La validación funcional se realizó en un entorno controlado; pruebas en campo abierto con condiciones heterogéneas y fluctuantes son necesarias para asegurar la robustez.

Estas limitaciones no disminuyen la relevancia del sistema, sino que más bien marcan un camino claro para el perfeccionamiento y adaptación a las exigencias reales del sector agrícola.

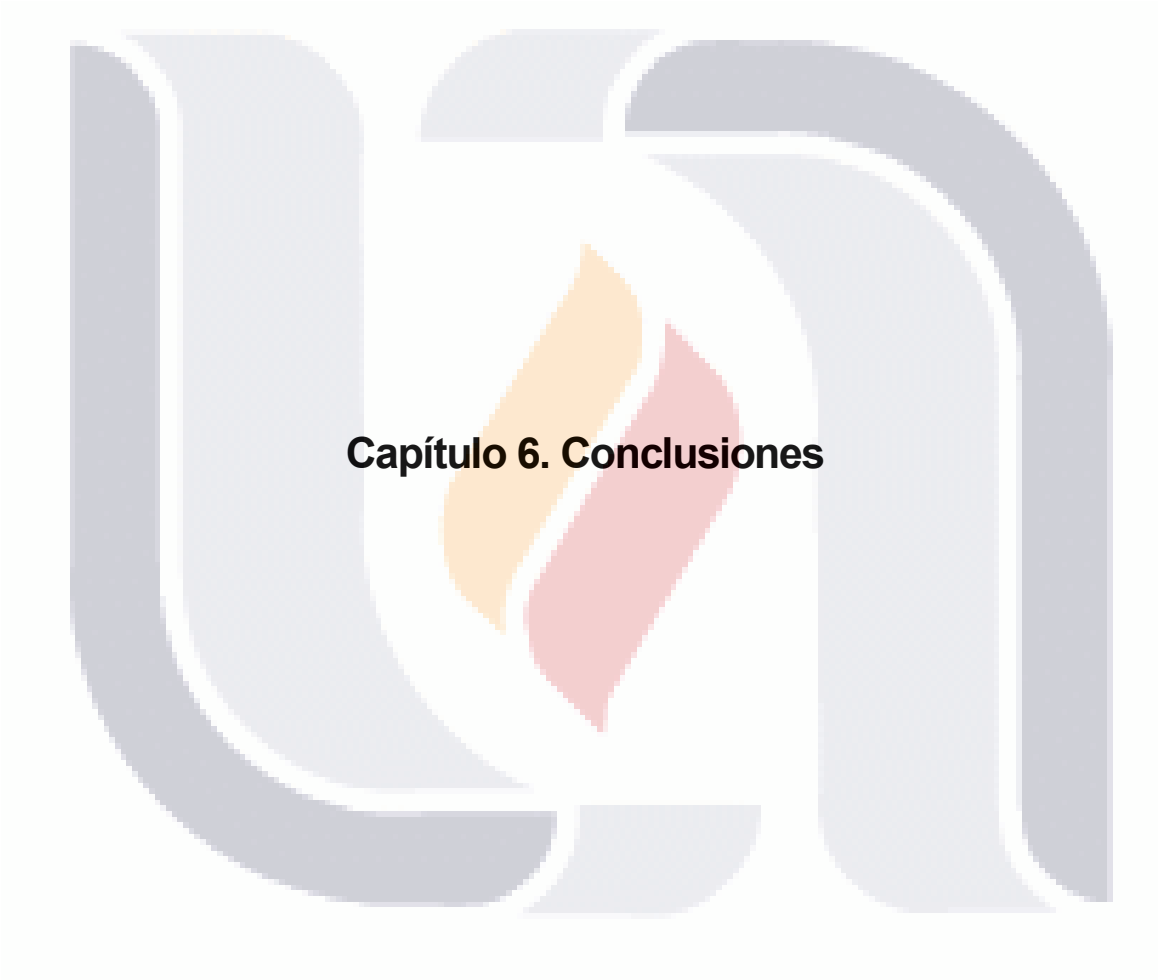
5.4.5. Contribuciones y perspectivas de impacto

Este proyecto contribuye significativamente a la agricultura inteligente, proponiendo un sistema integral que combina la recolección sensorial en tiempo real, la inteligencia predictiva y el control automatizado del riego. Sus aportes incluyen:

- Demostrar la factibilidad técnica y operativa de sistemas embebidos IoT aplicados a la gestión hídrica en cultivos protegidos.
- Proveer un modelo de predicción de humedad accesible y efectivo, que puede evolucionar con datos reales.

- Facilitar la adopción de tecnologías digitales en el sector agrícola, con un enfoque en sostenibilidad y optimización de recursos.
- Sentar las bases para desarrollos futuros que incorporen nuevas variables, modelos avanzados y ampliaciones a mayor escala.

El SIRCA-IoT no solo representa un avance tecnológico, sino una herramienta potencialmente transformadora para mejorar la productividad agrícola y reducir el impacto ambiental.



Capítulo 6. Conclusiones

6. Conclusiones

Este capítulo presenta las conclusiones generales del trabajo de investigación y desarrollo del SIRCA-IoT. Las conclusiones han sido estructuradas con base en el cumplimiento de los objetivos específicos planteados al inicio del proyecto, así como en los resultados obtenidos durante la implementación, validación y evaluación del sistema propuesto.

En este cierre, se reflexiona críticamente sobre la viabilidad técnica, operativa y funcional del sistema, su impacto potencial en el sector agrícola, y las contribuciones académicas que derivan de esta experiencia. Además, se exponen de manera detallada las limitaciones detectadas y las oportunidades de mejora identificadas a lo largo del proceso.

Finalmente, se presentan dos secciones complementarias: Trabajos Futuros, donde se describen las líneas de investigación y desarrollo que pueden derivarse de este trabajo, y Recomendaciones Finales, que orientan la implementación práctica del sistema en escenarios reales y sugieren buenas prácticas para su adopción efectiva y sostenible.

Estas conclusiones buscan no solo sintetizar lo realizado, sino también proyectar el alcance y la relevancia de la propuesta desarrollada, abriendo el camino hacia futuras aplicaciones en el ámbito de la agricultura inteligente y la gestión sostenible de los recursos hídricos.

6.1. Cumplimiento de los objetivos propuestos

El presente trabajo de tesis logró cumplir los objetivos propuestos a través del desarrollo e implementación del SIRCA-IoT, evidenciando una correspondencia directa entre los propósitos iniciales y los resultados alcanzados. A continuación, se presenta la trazabilidad explícita entre cada objetivo y las evidencias obtenidas, con referencia a las secciones donde se documenta su verificación.

- **Objetivo general (OG):**

  Desarrollar e implementar un sistema de riego inteligente basado en IoT que permita optimizar el consumo de agua en cultivos protegidos en Aguascalientes, México, mediante la recolección y análisis de datos ambientales, la predicción del nivel de humedad del suelo y el control automatizado del riego.

  Cumplimiento:

  Se desarrolló un prototipo funcional e integrado que combinó sensorización IoT (Raspberry Pi + DHT22, DS18B20, BH1750, SCD30), backend en Django/DRF con almacenamiento en TimescaleDB, frontend en Vue.js con visualización y control, y un modelo de machine learning para predicción de humedad. El sistema operó con latencia extremo a extremo promedio ≈ 1.6 s, estabilidad continuada > 12 h y desempeño predictivo $R^2 \approx 0.91$ con MAE ≈ 1.43 %, habilitando la toma de decisiones de riego automatizadas y fundamentadas en datos. (Véanse 4.2, 4.3, 4.4 y 4.2.5).

- **Objetivo específico 1 (OE1):** Integrar tecnología de IoT en el sistema de riego para monitorear variables críticas del entorno.

  Se logró porque:

  - Se conectaron e integraron sensores DHT22, DS18B20, BH1750 y SCD30 a Raspberry Pi mediante GPIO/I2C/1-Wire.
  - Se estableció publicación/subscripción MQTT por tópicos diferenciados y persistencia en TimescaleDB.
  - Se verificó flujo continuo y estable de datos durante pruebas prolongadas.

  (Secciones 4.2.1–4.2.3; validación en 4.2.5).

- **Objetivo específico 2 (OE2):** Diseñar y desarrollar una aplicación web que facilite el manejo del sistema de riego inteligente.

  Se logró porque:

- Se implementó backend Django/DRF con API REST y WebSockets para actualización en tiempo real.
- Se desarrolló frontend en Vue.js/Vuetify con panel de control, gráficos e histórico.
- Se habilitó control manual/supervisado de la bomba desde la interfaz.

(Secciones 4.3.1–4.3.3).

- **Objetivo específico 3 (OE3):** Incorporar un modelo predictivo de machine learning que sugiera el momento óptimo para activar el riego.

  Se logró porque:
  - Se entrenó e integró un modelo de regresión (Random Forest) con datos simulados de alta fidelidad.
  - Se obtuvo $R^2 \approx 0.91$ y MAE ≈ 1.43 %, con integración al backend para emitir sugerencias/activaciones.

  (Secciones 4.4.3–4.4.5).

- **Objetivo específico 4 (OE4):** Validar el sistema en un entorno de pruebas controlado.

  Se logró porque:
  - Se midió latencia total ≈ 1.6 s (adquisición → visualización/acción).
  - Se aseguró QoS 1, reconexión automática y heartbeat cada 5 min; operación estable > 12 h.
  - Se comprobó activación remota confiable de la bomba vía MQTT con confirmación de eventos.

  (Secciones 4.2.4–4.2.5).

- **Objetivo específico 5 (OE5):** Documentar detalladamente el proceso de diseño, desarrollo y validación del sistema.

  Se logró porque:

- Se elaboraron diagramas de arquitectura y flujo, esquemas eléctricos, bitácoras técnicas, código modular y reportes de pruebas con capturas e indicadores.

(Sección 4.2.5 y Anexos A–B).

Objetivo	Evidencias clave	Dónde se demuestra
OG	Integración IoT + web + ML; latencia ≈ 1.6 s; R^2 ≈ 0.91; MAE ≈ 1.43 %; operación > 12 h	4.2; 4.3; 4.4; 4.2.5
OE1	Sensores integrados; MQTT estable; almacenamiento en TimescaleDB	4.2.1–4.2.3; 4.2.5
OE2	API REST + WebSockets; UI con control de bomba e histórico	4.3.1–4.3.3
OE3	Random Forest integrado; métricas R^2/MAE; sugerencias/activaciones	4.4.3–4.4.5
OE4	Latencia total medida; QoS 1 y reconexión; activación remota confiable	4.2.4–4.2.5
OE5	Diagramas, bitácoras, código y resultados documentados	4.2.5; Anexos A–B

Tabla 8. Matriz de cumplimiento de objetivos.

6.2. Viabilidad técnica y operativa del sistema propuesto

La evaluación exhaustiva del sistema de riego inteligente desarrollado permitió confirmar su viabilidad técnica y operativa en escenarios controlados, sentando una base sólida para su futura implementación en entornos agrícolas reales. Esta viabilidad se manifiesta en múltiples dimensiones del sistema, incluyendo su arquitectura tecnológica, rendimiento funcional, estabilidad en el tiempo y capacidad de adaptación.

a) Integración funcional y sincronización operativa

Uno de los aspectos más relevantes que sustentan la viabilidad técnica es la integración completa y sincronizada de todos los componentes clave: sensores, nodos IoT, red de comunicación, backend de procesamiento, modelo predictivo de machine learning, interfaz de usuario y sistema de actuación. Esta integración logró establecer un flujo continuo de información desde la captura de datos en campo hasta la ejecución de acciones automáticas de riego, con mínima latencia y sin interrupciones operativas. El uso del protocolo MQTT con calidad de servicio (QoS) nivel 1 garantizó una transmisión confiable de datos, mientras que la arquitectura basada en microservicios y contenedores facilitó el despliegue distribuido de los servicios.

b) Rendimiento en tiempo casi real

Las pruebas realizadas revelaron que el sistema puede operar en tiempo casi real, con un tiempo promedio de respuesta total de aproximadamente 1.6 segundos desde la adquisición de datos hasta la activación física de la bomba de riego. Este rendimiento es adecuado para aplicaciones donde se requiere reaccionar oportunamente a cambios en las condiciones del suelo, lo cual es esencial para mantener la salud del cultivo y evitar el desperdicio de agua.

Además, el consumo de recursos computacionales fue bajo y constante, tanto en la Raspberry Pi como en el backend, permitiendo una operación eficiente en hardware de bajo costo y bajo consumo energético. Este aspecto es particularmente importante para entornos agrícolas que requieren soluciones accesibles y autónomas en términos energéticos.

c) Estabilidad y confiabilidad operativa

Durante pruebas extendidas de funcionamiento continuo, el sistema demostró una alta estabilidad, manteniendo la conexión a la red, evitando pérdidas de mensajes, y permitiendo la reconexión automática ante caídas breves de conexión. No se presentaron errores críticos ni fallos de sincronización, lo cual indica que el sistema puede sostener su operatividad durante largos períodos, incluso en condiciones variables o con conectividad intermitente, como suele ocurrir en zonas rurales.

d) Adaptabilidad y escalabilidad

El diseño modular y distribuido del sistema contribuye directamente a su viabilidad operativa a mayor escala, ya que permite incorporar nuevos nodos de monitoreo o módulos de control sin necesidad de rediseñar la arquitectura existente. La contenerización de los servicios backend y la eficiencia de la base de datos TimescaleDB en el manejo de series temporales facilitan la migración del sistema a entornos más complejos o productivos, como granjas de mayor tamaño o cultivos diversificados.

e) Interfaz de usuario y experiencia operativa

La interfaz gráfica desarrollada proporciona visualización clara y en tiempo real de las variables clave y los estados del sistema. Esta característica permite a los usuarios supervisar y entender fácilmente el funcionamiento del sistema, intervenir manualmente cuando sea necesario y confiar en el sistema para decisiones automatizadas fundamentadas en análisis predictivo.

En conjunto, todos estos factores demuestran que el sistema propuesto no solo es técnicamente factible, sino también operativamente sólido y sustentable, posicionándolo como una herramienta práctica, confiable y con un alto potencial para mejorar la eficiencia hídrica y tecnológica en la agricultura. La implementación futura en entornos reales requerirá adaptaciones menores, pero no compromete la solidez de la solución desarrollada.

6.3. Desempeño y utilidad del modelo predictivo

El modelo de *machine learning* integrado en el sistema de riego inteligente constituye uno de los componentes centrales para la automatización eficiente y el uso racional del recurso hídrico. Su desempeño fue evaluado desde una perspectiva cuantitativa (precisión estadística) y cualitativa (impacto en la toma de decisiones de riego), y los resultados obtenidos reflejan una utilidad práctica significativa y un comportamiento predictivo robusto, incluso en condiciones simuladas.

a) Precisión y capacidad de generalización

Durante el entrenamiento y validación del modelo, se alcanzó un coeficiente de determinación (R^2) de aproximadamente 0.91 y un error absoluto medio (MAE) cercano al 1.43 %, indicadores que evidencian una alta capacidad del modelo para predecir los niveles de humedad del suelo con exactitud. Esta precisión permitió identificar con fiabilidad situaciones críticas en las que el nivel de humedad desciende por debajo de los umbrales establecidos, lo cual habilita la activación preventiva del riego antes de que el déficit hídrico impacte negativamente en el cultivo.

A pesar de haber sido entrenado con datos simulados, el modelo demostró una consistencia interna robusta, lo cual valida su diseño algorítmico y lo posiciona como una base confiable para su futura adaptación con datos reales de campo.

b) Aporte a la eficiencia hídrica y toma de decisiones

El valor agregado del modelo no reside solamente en su precisión matemática, sino en su capacidad para influir positivamente en el proceso de toma de decisiones automatizadas, reduciendo la dependencia de criterios fijos o decisiones empíricas por parte de los operadores. En las pruebas funcionales, el modelo permitió:

- Reducir riegos innecesarios, evitando activaciones cuando la humedad del suelo se encontraba en niveles óptimos.
- Optimizar el uso del agua, activando el sistema únicamente cuando las condiciones reales y las predicciones futuras lo justificaban.
- Anticipar escenarios de déficit hídrico, con suficiente margen para responder antes de que se generen daños o estrés en el cultivo.

Estas capacidades no solo contribuyen a mejorar la sostenibilidad del sistema agrícola, sino que además posicionan el modelo como un elemento clave para una agricultura de precisión orientada a la conservación de recursos naturales.

c) Limitaciones actuales y perspectivas de mejora

Si bien el modelo demostró un desempeño notable, es importante subrayar ciertas limitaciones actuales que afectan su aplicabilidad inmediata en condiciones de campo:

- El entrenamiento con datos simulados, aunque útil en etapas iniciales, requiere ser complementado con datos reales y diversos que reflejen condiciones agroclimáticas locales.
- La ausencia de variables climáticas complementarias (precipitación, viento, radiación solar, entre otras) restringe la capacidad del modelo para capturar dinámicas complejas del entorno natural.
- La lógica de decisión se basa en umbrales estáticos, lo cual podría limitar la adaptabilidad ante variaciones abruptas del clima o condiciones del cultivo.

No obstante, estas limitaciones no comprometen el valor del modelo desarrollado. Más bien, ofrecen oportunidades claras para su evolución hacia esquemas más complejos, como el aprendizaje en línea (online learning), la incorporación de redes neuronales recurrentes (RNN) para modelar dependencias temporales, o el uso de sistemas híbridos que combinen reglas expertas y aprendizaje automático.

d) Utilidad como herramienta tecnológica

Desde una perspectiva aplicada, el modelo se consolida como una herramienta tecnológica útil, accesible y eficiente, capaz de integrarse en sistemas de bajo costo y operar en tiempo real. Su diseño modular, su bajo requerimiento computacional y su compatibilidad con sistemas embebidos como Raspberry Pi lo hacen apto para ser utilizado en contextos rurales con limitaciones de infraestructura, sin sacrificar precisión ni velocidad.

En resumen, el modelo predictivo no solo cumple su propósito técnico con un alto grado de precisión y utilidad, sino que además aporta inteligencia y adaptabilidad al sistema global, potenciando el impacto del riego automatizado y sentando las bases para el desarrollo de soluciones aún más avanzadas en el marco de la agricultura inteligente.

6.4. Aportes del sistema al sector agrícola

El desarrollo e implementación del sistema de riego inteligente propuesto representa una contribución significativa al proceso de modernización del sector agrícola, particularmente en contextos donde el acceso a tecnologías avanzadas aún es limitado. A través de una arquitectura integrada y modular que combina Internet de las Cosas (IoT), análisis predictivo mediante *machine learning* y automatización, el sistema aporta soluciones concretas a desafíos persistentes en la gestión del recurso hídrico, la eficiencia operativa y la toma de decisiones informadas.

a) Optimización del uso del agua

Uno de los aportes más relevantes del sistema es su capacidad para promover un uso más eficiente y racional del recurso hídrico, que es crítico en la agricultura moderna. Gracias a la monitorización en tiempo real de la humedad del suelo y a la integración de un modelo predictivo preciso, el sistema permite:

- Reducir el desperdicio de agua, evitando riegos innecesarios.
- Aplicar el riego únicamente cuando es necesario, con base en datos y pronósticos confiables.
- Mejorar la sostenibilidad de la producción agrícola, al minimizar el impacto ambiental derivado de la sobreirrigación o de prácticas empíricas de manejo del agua.

Este aporte es particularmente valioso en regiones con estrés hídrico, donde la disponibilidad del agua es limitada y su manejo eficiente resulta crucial para la seguridad alimentaria y la sostenibilidad económica.

b) Democratización del acceso a tecnologías digitales

El diseño del sistema, basado en hardware accesible como la Raspberry Pi, sensores de bajo costo y software libre, permite su adopción en entornos de bajos

recursos técnicos y económicos, eliminando barreras de entrada a la digitalización del agro. En este sentido, el sistema:

- Reduce la brecha tecnológica entre pequeños productores y grandes explotaciones agrícolas.
- Facilita la apropiación tecnológica por parte de usuarios sin formación técnica especializada, gracias a una interfaz intuitiva y a la automatización de procesos complejos.
- Incentiva la innovación local, al ser una plataforma abierta y escalable que puede adaptarse a las necesidades específicas de cada comunidad agrícola.

Este enfoque inclusivo y adaptable es clave para impulsar una transformación digital equitativa en el agro, especialmente en países en desarrollo.

c) Mejora en la toma de decisiones agronómicas

Al proporcionar información precisa, en tiempo real y visualmente accesible, el sistema facilita un enfoque de agricultura basada en datos, lo cual fortalece la capacidad de los productores para tomar decisiones más acertadas en cuanto al riego, el manejo del cultivo y la planificación de recursos. Esto se traduce en:

- Menor dependencia de la intuición o experiencia subjetiva del agricultor.
- Mayor capacidad para anticipar problemas y aplicar soluciones proactivas.
- Base sólida para integrar otras prácticas de agricultura de precisión, como fertilización localizada, monitoreo de plagas o análisis multivariable del entorno.

Esta transformación en la cultura de toma de decisiones representa un salto cualitativo hacia una gestión agrícola más científica, sostenible y productiva.

d) Plataforma para innovación y expansión

El sistema propuesto no solo resuelve una necesidad actual, sino que se proyecta como una plataforma versátil para futuras ampliaciones e innovaciones. Gracias a

su arquitectura modular y su compatibilidad con tecnologías modernas (contenedores, protocolos abiertos, bases de datos escalables), es posible:

- Incorporar nuevas variables ambientales, como precipitación, radiación solar o velocidad del viento.
- Ampliar el modelo predictivo a otras variables agronómicas, como el crecimiento del cultivo o la detección de enfermedades.
- Integrar el sistema a plataformas mayores, como sistemas de gestión agrícola (Farm Management Systems), redes de sensores distribuidos o infraestructuras en la nube.

Esto habilita su adopción en explotaciones agrícolas de diversas escalas y características, desde invernaderos familiares hasta grandes fincas tecnificadas, adaptándose a los requerimientos específicos de cada contexto productivo.

e) Contribución al desarrollo sostenible

Finalmente, el sistema aporta de manera directa a los Objetivos de Desarrollo Sostenible (ODS), especialmente en los siguientes puntos:

- ODS 2: Hambre cero, al contribuir a una producción agrícola más eficiente y resiliente.
- ODS 6: Agua limpia y saneamiento, al promover el uso eficiente del agua.
- ODS 9: Industria, innovación e infraestructura, mediante la aplicación de tecnologías emergentes en el entorno rural.
- ODS 13: Acción por el clima, al reducir el uso excesivo de agua y su impacto ambiental.

En conjunto, estos aportes posicionan al sistema desarrollado como una herramienta de alto valor estratégico para el sector agrícola, no solo por sus capacidades técnicas inmediatas, sino también por su potencial transformador en términos de sostenibilidad, equidad tecnológica e innovación continua.

6.5. Limitaciones del trabajo

A pesar de los resultados satisfactorios alcanzados durante el desarrollo e implementación del sistema de riego inteligente basado en tecnologías IoT y modelos predictivos de *machine learning*, es fundamental reconocer una serie de limitaciones que condicionan tanto la validez externa de los hallazgos como el alcance práctico del sistema en entornos reales. Estas limitaciones no desmeritan el valor del trabajo, sino que identifican con claridad los aspectos que deben ser abordados en investigaciones o desarrollos futuros para robustecer la solución y facilitar su adopción a gran escala.

a) Uso de datos simulados para el entrenamiento del modelo

Una de las principales restricciones metodológicas fue la dependencia de datos simulados para entrenar y validar el modelo de predicción de humedad del suelo. Si bien estos datos fueron diseñados para representar condiciones razonablemente realistas, presentan las siguientes limitaciones:

- Falta de ruido e irregularidades típicas de los entornos reales, como lecturas erráticas de sensores o condiciones climáticas imprevistas.
- Ausencia de estacionalidad y variabilidad geográfica, lo que limita la capacidad del modelo para generalizar a otros contextos agroclimáticos.

En consecuencia, la eficacia del modelo en situaciones reales aún debe ser verificada, ajustada y reentrenada con datos obtenidos directamente del campo.

b) Limitación en la variedad de variables ambientales

Durante el diseño y evaluación del sistema se consideraron principalmente variables relacionadas con la humedad del suelo, temperatura y otros factores básicos. No obstante, la exclusión de variables ambientales clave —como la precipitación, radiación solar, velocidad del viento o evapotranspiración— reduce la capacidad del modelo para capturar con mayor precisión la dinámica hídrica del suelo. Esta omisión se traduce en:

- Menor sensibilidad del sistema a eventos meteorológicos críticos, como lluvias súbitas o periodos de sequía intensa.
- Riesgo de sobreestimación o subestimación de la necesidad de riego, especialmente en escenarios climáticos complejos o de transición.

Para lograr un sistema más robusto y adaptable, será indispensable incorporar sensores adicionales o conectividad con fuentes de datos meteorológicos externas.

c) Uso de lógica de control basada en umbrales fijos

La lógica actual de activación del sistema de riego se basa en umbrales estáticos predefinidos, lo cual, aunque funcional en entornos controlados, presenta ciertas limitaciones:

- Falta de adaptabilidad ante cambios contextuales, como el tipo de cultivo, la fase fenológica o condiciones meteorológicas cambiantes.
- Riesgo de decisiones subóptimas, cuando los valores umbral no reflejan adecuadamente la realidad específica del entorno productivo.

Una alternativa futura es implementar mecanismos de aprendizaje en línea o lógica difusa, que permitan ajustar automáticamente los umbrales en función de patrones de comportamiento histórico o de nuevas observaciones en campo.

d) Validación en un entorno controlado

Todas las pruebas funcionales y experimentales fueron realizadas en un entorno de laboratorio o simulado, lo cual garantiza condiciones estables y repetibles, pero limita la validez externa de los resultados. Específicamente:

- No se evaluó la respuesta del sistema ante condiciones impredecibles o extremas, como desconexiones prolongadas, fallos eléctricos o eventos climáticos abruptos.
- No se consideraron aspectos logísticos y operativos del uso en campo abierto, como la exposición prolongada de los dispositivos, la interferencia con actividades humanas o animales, o los desafíos de mantenimiento.

La implementación en ambientes agrícolas reales será un paso imprescindible para determinar la resiliencia, escalabilidad y adopción práctica del sistema.

e) Seguridad y protección de datos

Si bien se establecieron mecanismos básicos de comunicación confiable mediante MQTT con calidad de servicio (QoS 1) y reconexión automática, no se implementaron medidas avanzadas de seguridad como:

- Cifrado de extremo a extremo (por ejemplo, TLS/SSL) para asegurar la privacidad de los datos transmitidos.
- Autenticación robusta para prevenir accesos no autorizados al sistema o a la base de datos.
- Políticas de respaldo y recuperación ante fallos, que garanticen la disponibilidad y consistencia de la información almacenada.

Estas omisiones pueden suponer riesgos significativos en contextos productivos reales donde la integridad y confidencialidad de los datos son críticas.

En resumen, aunque el sistema ha demostrado su funcionalidad y potencial en escenarios controlados, estas limitaciones constituyen áreas de mejora clave para futuras versiones. Superarlas permitirá aumentar la confiabilidad, flexibilidad y aplicabilidad del sistema en condiciones agrícolas reales, y fortalecerá su potencial como herramienta de transformación tecnológica en el agro.

6.6. Escalabilidad y perspectivas de mejora

El sistema desarrollado no solo ha demostrado ser técnicamente viable y funcional en un entorno controlado, sino que también presenta una arquitectura favorable para su escalabilidad y evolución futura, tanto a nivel técnico como operativo. Esta sección expone el potencial de crecimiento del sistema y plantea las principales líneas de mejora que podrían fortalecer su utilidad y adopción en escenarios agrícolas reales y de mayor complejidad.

a) Escalabilidad horizontal del sistema

Uno de los pilares del diseño propuesto es su arquitectura modular y distribuida, lo cual permite una escalabilidad horizontal efectiva. Específicamente:

- Se pueden agregar múltiples nodos IoT (Raspberry Pi) en distintas zonas de un cultivo o en diferentes parcelas, sin necesidad de modificar la arquitectura central del backend ni la lógica general del sistema.
- Cada nodo opera de forma autónoma y se comunica mediante MQTT con el servidor central, lo que minimiza los acoplamientos y facilita el mantenimiento o expansión del sistema.
- La base de datos TimescaleDB, optimizada para series temporales, puede gestionar volúmenes crecientes de datos sin comprometer el rendimiento, lo que posibilita la integración de cientos o miles de sensores en implementaciones a gran escala.

Este enfoque distribuye la carga de trabajo, evita cuellos de botella y permite adaptarse a explotaciones agrícolas de mayor tamaño o incluso a redes de cultivos geográficamente dispersos.

b) Contenerización y portabilidad del sistema

El uso de tecnologías como Docker para contenerizar los servicios del backend (API, base de datos y lógica de control) proporciona un entorno portátil y reproducible, lo que facilita el despliegue en distintos escenarios, tales como:

- Servidores locales en granjas o invernaderos, aprovechando infraestructuras existentes.
- Plataformas en la nube (AWS, Azure, Google Cloud, etc.), que permiten escalar automáticamente según la carga y necesidades.
- Dispositivos edge con capacidades intermedias, para una computación más cercana al origen de los datos.

Esto reduce las barreras para la adopción del sistema en ambientes con distintas capacidades técnicas y presupuestarias, y permite una rápida replicación del entorno para pruebas, actualizaciones o mantenimiento.

c) Integración de nuevas funcionalidades y tecnologías

La arquitectura modular del sistema facilita la incorporación futura de nuevas funcionalidades, tales como:

- Sensores adicionales para variables como radiación solar, velocidad del viento, pH o salinidad del suelo.
- Modelos de machine learning más complejos, incluyendo redes neuronales profundas o sistemas de aprendizaje en línea adaptativo.
- Lógica de control dinámica basada en pronósticos climáticos, modelos agronómicos o algoritmos de optimización multiobjetivo.
- Sistemas de alerta inteligentes que notifiquen al usuario en tiempo real a través de diferentes canales (SMS, correo, app móvil).

Asimismo, se puede considerar la incorporación de paneles solares y sistemas de bajo consumo energético, que harían viable la operación continua del sistema en zonas rurales con acceso eléctrico limitado.

d) Mejora de la robustez y seguridad del sistema

Desde el punto de vista operativo, existen oportunidades concretas para mejorar la resiliencia y la seguridad, incluyendo:

- Implementación de protocolos de cifrado TLS/SSL en las comunicaciones MQTT y REST para proteger la integridad y confidencialidad de los datos.
- Desarrollo de mecanismos de autenticación y control de acceso, que impidan el uso indebido del sistema o la manipulación de datos sensibles.
- Incorporación de módulos de respaldo y recuperación, que aseguren la continuidad operativa en caso de fallos o pérdida de conectividad.

Estas mejoras fortalecerían la confianza en el sistema y permitirían su adopción en ambientes productivos críticos donde la seguridad y disponibilidad de los datos son esenciales.

e) Adaptabilidad a diferentes tipos de cultivos y contextos productivos

El sistema puede evolucionar para ser más flexible y configurable, permitiendo su adaptación a distintos tipos de cultivo, condiciones edafoclimáticas y necesidades productivas. Para ello, se plantean las siguientes perspectivas:

- Desarrollo de interfaces de configuración agronómica, donde el usuario defina parámetros específicos según tipo de cultivo, fase fenológica o manejo del agua.
- Inclusión de módulos de aprendizaje autónomo, que ajusten dinámicamente los umbrales y comportamientos del sistema a partir de la experiencia acumulada.
- Integración con servicios externos de predicción climática y monitoreo satelital, ampliando el contexto de toma de decisiones.

Estas líneas de evolución permitirán que el sistema pase de una herramienta funcional a una plataforma inteligente adaptable, capaz de optimizar la gestión hídrica en diversas realidades agrícolas.

En síntesis, el sistema propuesto tiene un amplio potencial de escalabilidad y mejora, sustentado en decisiones tecnológicas adecuadas y una arquitectura preparada para el crecimiento. Estas características lo convierten en una solución prometedora no solo para su implementación inmediata en cultivos controlados, sino también para su transformación en una plataforma agrícola inteligente robusta, adaptable y de gran alcance.

6.7. Contribución académica y científica

El desarrollo de este sistema de riego inteligente basado en IoT y modelos predictivos de machine learning constituye una contribución significativa tanto en el ámbito académico como en el científico-tecnológico, al abordar de forma integral un problema crítico para la agricultura moderna: la gestión eficiente del agua. Este apartado presenta una reflexión detallada sobre los aportes generados en términos de conocimiento, metodologías y potencial de transferencia a la práctica.

a) Integración multidisciplinaria de conocimientos

Uno de los aportes más relevantes de este trabajo es su enfoque interdisciplinario, que articula conceptos y técnicas de diversas áreas del conocimiento, incluyendo:

- Ingeniería electrónica y de control, aplicada a la adquisición de datos mediante sensores y actuadores embebidos.
- Ciencias de la computación y software, mediante la implementación de arquitecturas backend, bases de datos temporales y protocolos de comunicación (MQTT).
- Machine learning, con la formulación, entrenamiento y evaluación de un modelo predictivo aplicado a variables ambientales.
- Agronomía y sostenibilidad, al orientar el sistema a la optimización del recurso hídrico y el incremento de la eficiencia agrícola.

Este enfoque holístico favorece la formación de profesionales capaces de abordar retos complejos desde múltiples dimensiones, y sienta un precedente para proyectos de investigación que requieran combinar hardware, software e inteligencia artificial con conocimientos del entorno productivo.

b) Generación de una metodología replicable

El trabajo propone y valida una metodología de desarrollo replicable para construir soluciones tecnológicas aplicadas a la agricultura de precisión, que incluye:

- La estructuración de un flujo completo desde la adquisición de datos hasta la toma de decisiones automatizada.
- El uso de datos simulados como estrategia preliminar de validación, lo cual permite iniciar desarrollos en ausencia de datos históricos reales, una situación común en muchas regiones agrícolas.
- La implementación de herramientas open source y tecnologías accesibles (Raspberry Pi, Django, Docker, Scikit-learn), que democratizan el acceso a soluciones avanzadas incluso en contextos con recursos limitados.

Esta metodología puede ser reutilizada, adaptada o extendida por futuros investigadores, tanto en el ámbito académico como en la industria tecnológica enfocada en el agro.

c) Aporte al estado del arte en agricultura inteligente

El sistema diseñado y evaluado se enmarca dentro del campo emergente de la agricultura inteligente (Smart Farming), y representa un avance en el estado del arte al proponer:

- Una solución completa, de extremo a extremo, que incluye sensores, análisis inteligente y actuadores.
- Un modelo predictivo funcional para la humedad del suelo, con resultados cuantitativos que evidencian su utilidad práctica ($R^2 \approx 0.91$ y MAE bajo 1.5 %).
- Una arquitectura tecnológica realista, adaptable y pensada para escenarios reales de uso, incluyendo capacidades de escalamiento, portabilidad y monitoreo remoto.

Estos elementos contribuyen a la literatura científica y técnica sobre sistemas ciberfísicos aplicados a la gestión del riego, y pueden ser base para publicaciones académicas, artículos científicos o desarrollos tecnológicos avanzados.

d) Potencial de transferencia y aplicación práctica

Más allá del ámbito académico, el sistema presenta un potencial real de transferencia tecnológica, al ofrecer una solución viable y adaptable para:

- Productores agrícolas medianos o pequeños que buscan mejorar la eficiencia en el uso del agua y reducir costos.
- Instituciones de investigación y extensión rural, que pueden utilizar esta plataforma como base para pruebas, capacitación y difusión tecnológica.
- Desarrolladores y startups agrotecnológicas, interesados en adaptar la solución a contextos específicos o escalarla como producto comercial.

Este trabajo, por tanto, no solo genera conocimiento teórico, sino que acerca la innovación tecnológica al terreno productivo, promoviendo una agricultura más inteligente, precisa y sustentable.

En conclusión, la presente tesis representa una aportación académica sólida y técnicamente fundamentada, que amplía las posibilidades de investigación, desarrollo y aplicación en el campo de la agricultura digital. Su contribución radica en haber construido y validado una solución concreta que, además de demostrar su factibilidad, ofrece caminos claros para la mejora, adaptación y ampliación del conocimiento en futuras iniciativas académicas, científicas y tecnológicas.

6.8. Reflexión final

La culminación de este trabajo representa no solo el cierre de una etapa académica, sino también el punto de partida para nuevas líneas de desarrollo, investigación y aplicación práctica en el ámbito de la agricultura inteligente. A lo largo del proceso de diseño, implementación y validación del sistema de riego automatizado basado en IoT y machine learning, fue posible constatar la capacidad de la tecnología para ofrecer soluciones concretas y sostenibles a problemas reales como la gestión eficiente del agua en la producción agrícola.

Esta experiencia puso de manifiesto la importancia de la innovación tecnológica aplicada con un enfoque contextual y ético, donde el objetivo no sea únicamente

optimizar procesos, sino también contribuir al bienestar de las comunidades, al cuidado del medio ambiente y a la sostenibilidad de los recursos naturales. En ese sentido, el sistema desarrollado no se limita a ser un ejercicio académico o un prototipo funcional, sino que constituye una propuesta con proyección social, económica y ecológica.

Desde una perspectiva formativa, este proyecto ha permitido fortalecer competencias técnicas y metodológicas en áreas clave como el diseño de sistemas embebidos, el manejo de bases de datos temporales, la analítica predictiva y la integración de arquitecturas distribuidas. Pero más allá del dominio técnico, se ha reafirmado también el valor de una visión integradora, crítica y propositiva, que articule ciencia, tecnología y contexto local para generar impactos positivos.

Finalmente, este trabajo invita a continuar profundizando en el campo de la agricultura de precisión, fomentando una mayor colaboración interdisciplinaria, el uso responsable de los datos y la búsqueda de soluciones adaptadas a los desafíos del presente y del futuro. La ruta hacia una agricultura más resiliente, eficiente y sustentable no depende únicamente de avances tecnológicos, sino también de la voluntad de aplicarlos con propósito y compromiso. En ese camino, esta tesis aspira a ser una contribución valiosa y una base sólida para nuevas exploraciones e innovaciones.

6.9. Trabajos Futuros

La culminación de este proyecto marca no solo un punto de cierre, sino también el inicio de múltiples líneas de continuidad que permitirán profundizar, robustecer y extender las capacidades del sistema de riego inteligente basado en IoT y machine learning. A partir de las pruebas realizadas, las limitaciones identificadas y el potencial tecnológico evidenciado, se proponen a continuación diversos frentes de investigación y desarrollo a considerar en trabajos futuros:

1) Validación en condiciones reales de campo

Uno de los pasos más relevantes hacia la consolidación del sistema es su validación en contextos agrícolas reales, fuera del entorno controlado de pruebas. Esta fase permitirá evaluar el comportamiento del sistema frente a condiciones ambientales no ideales, ruido sensorial, variabilidad edafológica y topográfica, así como factores operativos propios del trabajo agrícola. La implementación en campo abrirá la posibilidad de ajustar el modelo predictivo, mejorar la robustez del hardware y afinar los algoritmos de control, consolidando así su aplicabilidad práctica.

2) Inclusión de variables ambientales adicionales

El sistema actual se basa principalmente en mediciones de humedad del suelo, temperatura y datos básicos ambientales. Sin embargo, variables como precipitación pluvial, radiación solar, velocidad del viento, evapotranspiración y humedad relativa del aire pueden tener un impacto significativo en la dinámica hídrica del suelo. Su incorporación permitiría un análisis más completo y una mayor precisión en las predicciones del modelo, haciéndolo más resiliente frente a condiciones climáticas variables.

3) Desarrollo de modelos de aprendizaje adaptativo

El modelo actual opera sobre datos simulados y presenta un entrenamiento estático. Para mejorar su adaptabilidad, se propone investigar enfoques de aprendizaje automático en línea (online learning), que permitan al sistema actualizar sus parámetros de manera continua a medida que recopila nuevos datos. Esto incrementaría su capacidad de generalización y reduciría la necesidad de reentrenamientos manuales, lo cual es especialmente valioso en sistemas desplegados durante largos períodos o en entornos altamente dinámicos.

4) Integración con pronósticos meteorológicos

La conexión del sistema con servicios de pronóstico del clima, mediante APIs abiertas o servicios especializados, podría enriquecer el proceso de toma de decisiones. La combinación de datos sensoriales en tiempo real con predicciones meteorológicas ofrecería una visión prospectiva del estado hídrico del suelo y

permitiría anticipar necesidades de riego de forma más eficiente, evitando riegos innecesarios o inadecuados.

5) Mejora de la interfaz de usuario y experiencia de uso

Si bien la interfaz actual permite la visualización y control básico del sistema, se sugiere su rediseño bajo principios de experiencia de usuario (UX) para mejorar la navegabilidad, accesibilidad e interacción. Asimismo, se recomienda el desarrollo de una aplicación móvil multiplataforma con capacidades offline, que facilite el uso por parte de agricultores con baja conectividad o limitado acceso a infraestructura digital.

6) Expansión hacia arquitecturas multizona y multiusuario

La arquitectura del sistema es susceptible de ampliación para controlar múltiples zonas de riego con condiciones y cultivos distintos. Explorar esta dirección permitiría el desarrollo de soluciones escalables aplicables a fincas medianas o grandes. Además, incorporar funcionalidades multiusuario (con distintos niveles de acceso y control) facilitaría el uso en entornos colaborativos o empresariales.

7) Implementación de seguridad integral

En futuros desarrollos, se vuelve indispensable reforzar la seguridad del sistema para su implementación en entornos reales. Esto incluye cifrado de datos (por ejemplo, TLS en MQTT), autenticación robusta de dispositivos y usuarios, y monitoreo de eventos de seguridad. Estos mecanismos garantizarán la integridad y confidencialidad de los datos transmitidos y almacenados, reduciendo el riesgo de vulnerabilidades.

8) Evaluación del impacto económico y ambiental

Una línea de investigación futura con enfoque interdisciplinario es la evaluación del impacto económico y ambiental de la adopción del sistema. Estimar el ahorro de agua, la reducción de costos de operación agrícola y los beneficios ambientales podría aportar evidencia cuantitativa de su valor, facilitando su adopción por parte de instituciones y políticas públicas orientadas a la agricultura sostenible.

6.10.Recomendaciones Finales

Como cierre de esta investigación, se presentan una serie de recomendaciones orientadas tanto a la mejora continua del sistema desarrollado como a su implementación responsable en entornos agrícolas reales. Estas recomendaciones recogen aprendizajes derivados del proceso de diseño, prueba y análisis del sistema, y están dirigidas a investigadores, desarrolladores, técnicos agrícolas y tomadores de decisiones interesados en tecnologías de agricultura inteligente.

Priorizar la validación con datos reales y en campo abierto

Si bien el modelo predictivo demostró alta precisión con datos simulados, su aplicación en entornos reales requiere una fase de reentrenamiento y validación con datos obtenidos en campo. Es recomendable iniciar campañas de recolección sistemática de datos de humedad, temperatura, precipitaciones y otras variables en distintos tipos de suelo, cultivos y regiones climáticas, para robustecer el modelo y aumentar su capacidad de generalización.

Fortalecer la infraestructura de conectividad en zonas rurales

Para que la solución pueda ser adoptada ampliamente, es necesario garantizar la conectividad de red en las zonas agrícolas donde se desea implementar. Se recomienda explorar opciones de comunicación híbrida (Wi-Fi, redes móviles, LoRaWAN, entre otras) que se adapten a las condiciones del terreno, así como mecanismos de almacenamiento en caché en los nodos IoT ante interrupciones temporales de red.

Adoptar buenas prácticas de seguridad desde el diseño

La seguridad debe integrarse de forma transversal en todo el sistema, especialmente cuando se maneja información sensible o se actúa sobre infraestructura física. Se recomienda implementar en versiones futuras:

- Cifrado TLS para el protocolo MQTT.

- Autenticación segura de usuarios y dispositivos.
- Registros de auditoría y monitoreo de actividad del sistema. Estas medidas son fundamentales para proteger el sistema ante accesos no autorizados, sabotajes o vulnerabilidades.

Fomentar la capacitación técnica de los usuarios finales

Para asegurar el uso efectivo del sistema, es recomendable desarrollar materiales educativos, guías técnicas y talleres dirigidos a los usuarios finales, especialmente agricultores y técnicos rurales. Estos materiales deben cubrir aspectos como el mantenimiento de sensores, interpretación de alertas y gestión de datos, promoviendo la apropiación tecnológica desde una perspectiva práctica y accesible.

Promover la colaboración interdisciplinaria

El desarrollo y despliegue exitoso de sistemas agrícolas inteligentes requiere la convergencia de múltiples disciplinas: ingeniería, agronomía, meteorología, informática y economía. Se recomienda fomentar alianzas entre universidades, centros de investigación, cooperativas agrícolas y organismos gubernamentales para enriquecer el sistema con conocimiento contextual, garantizar su pertinencia y facilitar su transferencia tecnológica.

Establecer mecanismos de monitoreo y mejora continua

Finalmente, es fundamental que cualquier implementación del sistema en un entorno real esté acompañada de indicadores de desempeño técnico, impacto económico, ahorro de recursos hídricos y nivel de adopción por parte de los usuarios. Estos indicadores permitirán evaluar de forma continua la efectividad del sistema y orientar decisiones sobre su mantenimiento, mejora o expansión.

Capítulo 7. Bibliografía

7. Bibliografía

[1] O. S. Olivares, A. L. Burgos, J. S. Ramírez, and G. Bocco, “Valoración de la seguridad hídrica con enfoque de cuenca hidrográfica: Aplicación en cuencas rurales del Centro Occidente de México.,” *Journal of Latin American Geography*, vol. 18, no. 2, pp. 88-88–119, 2019, doi: 10.1353/lag.2019.0035.

[2] P. Asghari, A. M. Rahmani, and H. H. S. Javadi, “Internet of Things applications: A systematic review,” *Computer Networks*, vol. 148, pp. 241–261, 2019.

[3] Gobierno del Estado de Aguascalientes, “Plan de Desarrollo del Estado 2022-2027,” 2023.

[4] Presidencia de la República, “Plan Nacional de Desarrollo 2025 - 2030,” Feb. 2025, [Online]. Available: https://www.gob.mx/presidencia/documentos/plan-nacional-de-desarrollo-2025-2030-391771

[5] S. Khriji, D. El Houssaini, I. Kammoun, and O. Kanoun, “Precision Irrigation: An IoT-Enabled Wireless Sensor Network for Smart Irrigation Systems,” in *Women in Precision Agriculture: Technological breakthroughs, Challenges and Aspirations for a Prosperous and Sustainable Future*, T. K. Hamrita, Ed., Cham: Springer International Publishing, 2021, pp. 107–129. doi: 10.1007/978-3-030-49244-1_6.

[6] F. M. Padilla, M. Farneselli, G. Gianquinto, F. Tei, and R. B. Thompson, “Monitoring nitrogen status of vegetable crops and soils for optimal nitrogen management,” *Agric Water Manag*, vol. 241, p. 106356, 2020, doi: https://doi.org/10.1016/j.agwat.2020.106356.

[7] R. Allan, L. Pereira, and M. Smith, *Crop evapotranspiration-Guidelines for computing crop water requirements-FAO Irrigation and drainage paper 56*, vol. 56. 1998.

[8] T. A. Howell, “Enhancing Water Use Efficiency in Irrigated Agriculture,” *Agron J*, vol. 93, no. 2, pp. 281–289, 2001, doi: https://doi.org/10.2134/agronj2001.932281x.

[9] S. Zhang, X. Wu, Z. You, and L. Zhang, “Leaf image based cucumber disease recognition using sparse representation classification,” *Comput Electron Agric*, vol. 134, pp. 135–141, Mar. 2017, doi: 10.1016/j.compag.2017.01.014.

[10] FAO, *The State of Food and Agriculture 2024*. Rome, Italy: FAO, 2024. doi: 10.4060/cd2616en.

[11] ITU-T, “ITU-T Rec. Y.2060 (06/2012) Overview of the Internet of things,” 2012.

[12] W. Tao, L. Zhao, G. Wang, and R. Liang, “Review of the internet of things communication technologies in smart agriculture and challenges,” Oct. 2021, *Elsevier B.V.* doi: 10.1016/j.compag.2021.106352.

[13] X. Ding and W. Du, “Optimizing Irrigation Efficiency using Deep Reinforcement Learning in the Field,” 2023. [Online]. Available: https://arxiv.org/abs/2304.01435

[14] R. A. Osman, “Optimizing IoT communication for enhanced data transmission in smart farming ecosystems,” *Expert Syst Appl*, vol. 265, Mar. 2025, doi: 10.1016/j.eswa.2024.125879.

[15] J. Omara, E. Talavera, D. Otim, D. Turcza, E. Ofumbi, and G. Owomugisha, “A field-based recommender system for crop disease detection using machine learning,” *Front Artif Intell*, vol. 6, 2023, doi: 10.3389/FRAI.2023.1010804/PDF.

[16] R. Strong, J. T. Wynn, J. R. Lindner, and K. Palmer, “Evaluating Brazilian Agriculturalists’ IoT Smart Agriculture Adoption Barriers: Understanding Stakeholder Salience Prior to Launching an Innovation,” *Sensors (Basel)*, vol. 22, no. 18, Sep. 2022, doi: 10.3390/S22186833.

[17] A. S. Albahri *et al.*, “Based Multiple Heterogeneous Wearable Sensors: A Smart Real-Time Health Monitoring Structured for Hospitals Distributor,” *IEEE Access*, vol. 7, pp. 37269–37323, 2019, doi: 10.1109/ACCESS.2019.2898214.

[18] L. Gałęzewski *et al.*, “Analysis of the need for soil moisture, salinity and temperature sensing in agriculture: a case study in Poland,” *Sci Rep*, vol. 11, no. 1, Dec. 2021, doi: 10.1038/S41598-021-96182-1.

[19] V. Aliabadi, S. Gholamrezai, and P. Ataei, “Rural people’s intention to adopt sustainable water management by rainwater harvesting practices: application of TPB and HBM models,” *Water supply*, vol. 20, no. 5, pp. 1847–1861, Aug. 2020, doi: 10.2166/WS.2020.094.

[20] S. J. Van De Meene, R. R. Brown, and M. A. Farrelly, “Capacity attributes of future urban water management regimes: projections from Australian sustainability practitioners.,” *Water Sci Technol*, vol. 61 9, no. 9, pp. 2241–50, 2010, doi: 10.2166/WST.2010.154.

[21] D. R. Marlow, D. J. Beale, and S. Burn, “A pathway to a more sustainable water sector: sustainability-based asset management.,” *Water Sci Technol*, vol. 61 5, no. 5, pp. 1245–55, 2010, doi: 10.2166/WST.2010.043.

[22] R. I. McDonald *et al.*, “Global Urban Growth and the Geography of Water Availability, Quality, and Delivery,” *Ambio*, vol. 40, no. 5, pp. 437–446, Jun. 2011, doi: 10.1007/S13280-011-0152-6.

[23] G. Salmoral *et al.*, “Water-related challenges in nexus governance for sustainable development: Insights from the city of Arequipa, Peru.,” *Sci Total Environ*, vol. 747, Dec. 2020, doi: 10.1016/J.SCITOTENV.2020.141114.

[24] S. A. Noorhosseini, M. S. Allahyari, C. A. Damalas, and S. S. Moghaddam, “Public environmental awareness of water pollution from urban growth: The case of Zarjub and Goharrud rivers in Rasht, Iran.,” *Sci Total Environ*, vol. 599–600, pp. 2019–2025, Dec. 2017, doi: 10.1016/J.SCITOTENV.2017.05.128.

[25] K. S. Fielding, A. Spinks, S. Russell, R. McCrea, R. Stewart, and J. Gardner, “An experimental test of voluntary strategies to promote urban water demand management.,” *J Environ Manage*, vol. 114, pp. 343–51, Jan. 2013, doi: 10.1016/J.JENVMAN.2012.10.027.

[26] S. Jayashree, M. N. H. Reza, C. A. N. Malarvizhi, and M. Mohiuddin, “Industry 4.0 implementation and Triple Bottom Line sustainability: An empirical study on small and medium manufacturing firms,” *Heliyon*, vol. 7, no. 8, Aug. 2021, doi: 10.1016/J.HELIYON.2021.E07753.

[27] R. Črešnar, V. Potočan, and Z. Nedelko, “Speeding Up the Implementation of Industry 4.0 with Management Tools: Empirical Investigations in Manufacturing Organizations,” *Sensors (Basel)*, vol. 20, no. 12, pp. 1–25, Jun. 2020, doi: 10.3390/S20123469.

[28] P. Strickland and K. M. Williams, “The adoption of smart industry 4.0 app technology and harnessing e-WOM in the wine industry caused by a global pandemic: a case study of the Yarra Valley in Australia,” *Journal of Hospitality and Tourism Insights*, 2022, doi: 10.1108/JHTI-05-2022-0175.

[29] S. Khin and D. M. H. Kee, “Factors influencing Industry 4.0 adoption,” *Journal of Manufacturing Technology Management*, vol. 33, no. 3, pp. 448–467, Mar. 2022, doi: 10.1108/JMTM-03-2021-0111.

[30] S. Chatterjee, R. Chaudhari, and R. Shams, “Applications of Industry 4.0 for Pandemic Responses and Business Continuity: A TOE-DCV Integrated Approach,” *IEEE Trans Eng Manag*, 2023, doi: 10.1109/TEM.2023.3250587.

[31] T. Christie, “Django REST Framework,” https://www.django-rest-framework.org/.

[32] Django Software Foundation, “Django Channels,” https://channels.readthedocs.io/en/latest/.

[33] Redis, “Redis - The Real-time Data Platform,” https://redis.io/.

[34] A. Solem, “Celery - Distributed Task Queue,” https://docs.celeryq.dev/en/stable/.

[35] E. You, “Vue.js,” https://vuejs.org/.

[36] Vuetify, “Vuetify — A Vue Component Framework,” https://vuetifyjs.com/en/.

[37] ApexCharts, “ApexCharts.js - Open Source JavaScript Charts for your website,” https://apexcharts.com/.

[38] E. San Martin Morote, “PiniaThe intuitive store for Vue.js,” https://pinia.vuejs.org/.

[39] The Axios Project, “AXIOS,” https://axios-http.com/es/.

[40] Timescale Inc., “PostgreSQL ++ for time series and events | Timescale,” https://www.timescale.com/.

[41] The PostgreSQL Global Development Group, “PostgreSQL,” https://www.postgresql.org/.

[42] L. V. S. Kumar and T. S. L. V Ayyarao, “Real Time Environmental Monitoring with Raspberry Pi 3B+ and MATLAB for Enhanced Worker Safety Through IoT,” Sep. 26, 2024, *IEEE*. doi: 10.1109/ICPEEV63032.2024.10931895.

[43] R. Zwetsloot, “Raspberry Pi 4 specs and benchmarks — Raspberry Pi Official Magazine,” https://magazine.raspberrypi.com/articles/raspberry-pi-4-specs-benchmarks.

[44] R. Shaik, F. Syed, K. Ratnam, and C. Bhargavi, “IoT based automated irrigation system using Raspberry Pi,” *International Journal of Electrical Engineering and Technology*, vol. 11, no. 3, 2020.

[45] A. Dawod, D. Georgakopoulos, P. P. Jayaraman, and A. Nirmalathas, “A Survey of Techniques for Discovering, Using, and Paying for Third-Party IoT Sensors.,” *Sensors (14248220)*, vol. 24, no. 8, p. 2539, Apr. 2024, doi: 10.3390/s24082539.

[46] P. Cihan, “IoT Technology in Smart Agriculture,” *International Conference on Recent Academic Studies*, vol. 1, pp. 185–192, May 2023, doi: 10.59287/icras.693.

[47] Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and and Rahul Gupta, “MQTT Version 5.0,” Mar. 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[48] HiveMQ, “HiveMQ – The Most Trusted MQTT platform to Transform Your Business,” https://www.hivemq.com/.

[49] Django Software Foundation, “Django documentation,” https://docs.djangoproject.com/en/5.2/.

[50] A. Saif Uddin and A. Solem, “django-celery-beat - Database-backed Periodic Tasks,” https://django-celery-beat.readthedocs.io/en/latest/.

[51] N. Liyanage, C. Attanayaka, T. Perera, D. Neilkumara, I. S. Bandara, and L. Chandrasiri, “IoT-Based Smart Beehive Monitoring System,” Dec. 12, 2024, *IEEE*. doi: 10.1109/ICAC64487.2024.10851052.

[52] M. Del-Coco, M. Leo, and P. Carcagnì, “Machine Learning for Smart Irrigation in Agriculture: How Far along Are We?,” *Information*, vol. 15, no. 6, 2024, doi: 10.3390/info15060306.

[53] L. Umutoni and V. Samadi, “Application of machine learning approaches in supporting irrigation decision making: A review,” *Agric Water Manag*, vol. 294, p. 108710, 2024, doi: https://doi.org/10.1016/j.agwat.2024.108710.

[54] R. Togneri *et al.*, “Soil moisture forecast for smart irrigation: The primetime for machine learning,” *Expert Syst Appl*, vol. 207, p. 117653, 2022, doi: https://doi.org/10.1016/j.eswa.2022.117653.

[55] Y. Wang, L. Shi, Y. Hu, X. Hu, W. Song, and L. Wang, “A comprehensive study of deep learning for soil moisture prediction,” *Hydrol Earth Syst Sci*, vol. 28, no. 4, pp. 917–943, 2024, doi: 10.5194/hess-28-917-2024.

[56] L. P. Challa, C. D. Singh, K. V. R. Rao, A. Subeesh, and M. Srilakshmi, “Prediction of soil moisture using machine learning techniques: A case study of an IoT-based irrigation system in a naturally ventilated polyhouse,” *Irrigation*

and Drainage, vol. 73, no. 3, pp. 1138–1150, Jul. 2024, doi: https://doi.org/10.1002/ird.2933.

[57] M. Taheri, M. Bigdeli, H. Imanian, and A. Mohammadian, “An Overview of Machine-Learning Methods for Soil Moisture Estimation,” *Water (Basel)*, vol. 17, no. 11, 2025, doi: 10.3390/w17111638.

[58] N. Ghadiri, B. Javadi, O. Obst, and S. Pfautsch, “Data Optimisation of Machine Learning Models for Smart Irrigation in Urban Parks,” in *2024 International Conference on Ubiquitous Computing and Communications (IUCC)*, IEEE, Dec. 2024, pp. 70–77. doi: 10.1109/iucc65928.2024.00012.

[59] scikit-learn, “Cross-validation: evaluating estimator performance,” https://scikit-learn.org/stable/modules/cross_validation.html?utm_source=chatgpt.com.

[60] M. Bhagat and B. Bakariya, “A Comprehensive Review of Cross-Validation Techniques in Machine Learning,” Jan. 2025. doi: 10.71097/IJSAT.v16.i1.1305.

[61] X. Ding and W. Du, “Optimizing Irrigation Efficiency using Deep Reinforcement Learning in the Field,” 2023. [Online]. Available: https://arxiv.org/abs/2304.01435

[62] A. Vyas and S. Bandyopadhyay, “Dynamic Structure Learning through Graph Neural Network for Forecasting Soil Moisture in Precision Agriculture,” 2022. [Online]. Available: https://arxiv.org/abs/2012.03506

[63] Raspberry Pi Foundation, “Raspberry Pi Documentation,” https://www.raspberrypi.com/documentation/.

[64] Python Software Foundation, “Paho MQTT Documentation.” [Online]. Available: https://www.eclipse.org/paho/

[65] I. Craggs, “Eclipse Paho | The Eclipse Foundation,” https://eclipse.dev/paho/.

[66] Lou. Hattersley, “The official Raspberry Pi handbook 2024,” p. 201, 2024.

[67] Y. G. Kusuma, S. Parvathi, Y. Sirisha, and Ch. Lasya, “Gas Detection and Environmental Monitoring Using Raspberry Pi Pico,” Jun. 21, 2024, *IEEE*. doi: 10.1109/SCES61914.2024.10652582.

Capítulo 9. Anexos

8. Anexos

Anexo A. Configuraciones de hardware y software

A.1 Instalación del sistema operativo y entorno de desarrollo

Instalación del Sistema Operativo y Actualización

Se recomienda utilizar Raspberry Pi OS (64-bit). Si aún no está instalado, se puede descargar desde la página oficial de Raspberry Pi y flashear en una tarjeta microSD utilizando Raspberry Pi Imager o balenaEtcher.

Actualizar el sistema operativo para garantizar que los paquetes están actualizados:

`sudo apt update && sudo apt upgrade -y` [63]

Instalación de Python y Configuración del Entorno Virtual

Python 3 viene preinstalado en Raspberry Pi OS, pero es recomendable asegurarse de que se tenga la versión más reciente e instalar las bibliotecas necesarias.

Instalar Python y pip:

```
sudo apt install python3 python3-pip -y
```

Crear y activar un entorno virtual para aislar las dependencias:

```
python3 -m venv .venv
```

`source .venv/bin/activate` [63]

Instalación de Bibliotecas Necesarias

Instalar las bibliotecas específicas utilizadas en este proyecto:

```
pip install Adafruit-Blinka adafruit-circuitpython-ads1x15 adafruit-circuitpython-bh1750 \
adafruit-circuitpython-busdevice adafruit-circuitpython-connectionmanager adafruit-circuitpython-dht \
adafruit-circuitpython-register adafruit-circuitpython-requests adafruit-circuitpython-scd4x \
adafruit-circuitpython-typing adafruit-io Adafruit-PlatformDetect Adafruit-PureIO binho-host-adapter \
certifi charset-normalizer click idna paho-mqtt pyftdi pyserial python-dotenv pyusb requests RPi.GPIO \
rpi_ws281x smbus2 sysv_ipc typing_extensions urllib3 w1thermsensor
```

Configuración de Buses I²C Adicionales

En este proyecto, se han creado dos buses I²C adicionales para manejar los tres sensores I²C utilizados. Para configurarlos, se debe modificar el archivo de configuración:

```
sudo nano /boot/firmware/config.txt
```

Agregar las siguientes líneas al final del archivo:

```
dtoverlay=i2c-gpio,bus=3,i2c_gpio_sda=23,i2c_gpio_scl=24
dtoverlay=i2c-gpio,bus=4,i2c_gpio_sda=27,i2c_gpio_scl=22
```

Guardar los cambios y reiniciar la Raspberry Pi:

```
sudo reboot
```

Para verificar los buses creados:

```
ls /dev/i2c-*
```

Debe mostrar algo como:

/dev/i2c-1 /dev/i2c-3 /dev/i2c-4 [44]

A.2 Verificación de sensores y configuración MQTT

Habilitación de Interfaces de Hardware

Para permitir la comunicación con los sensores, se deben habilitar las interfaces necesarias:

```
sudo raspi-config
```

Dentro del menú de configuración, habilitar en Interfacing Options:

- I²C (para sensores BH1750, SCD41 y ADS1115)
- 1-Wire (para el sensor DS18B20)
- SPI (si se requiere en futuras expansiones)

Después de realizar los cambios, reiniciar la Raspberry Pi:

sudo reboot [63]

Verificación de la Comunicación con los Sensores

Después de configurar los buses, se deben detectar los sensores conectados ejecutando:

```
sudo i2cdetect -y 3
```

```
sudo i2cdetect -y 4
```

Esto listará las direcciones de los sensores conectados en cada bus [63].

Configuración del Cliente MQTT

El sistema usa el protocolo MQTT para enviar y recibir datos de sensores y órdenes para la bomba de agua. Se debe instalar la librería paho-mqtt y verificar la conexión con el broker de HiveMQ.

Ejemplo de código para probar la conexión con HiveMQ:

```python
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conexión exitosa al broker MQTT")
    else:
        print(f"Error de conexión con código {rc}")

client = mqtt.Client()
client.on_connect = on_connect
client.connect("broker.hivemq.com", 1883, 60)
client.loop_start()
```

Código 21. Ejemplo de código para probar la conexión con HiveMQ.

Si el mensaje "Conexión exitosa al broker MQTT" aparece en la terminal, la configuración ha sido correcta.

Con esta configuración, la Raspberry Pi está lista para la adquisición de datos desde los sensores, el control de la bomba de agua y la transmisión de información mediante MQTT [64], [65].

A.3. Esquema de Conexión de Sensores y Actuadores

Conexión del DHT22 (Humedad y temperatura ambiental)

- Alimentación: 3.3V o 5V
- Comunicación: Digital (protocolo de un solo cable)
- Pines de conexión:
    - VCC → 3.3V o 5V de la Raspberry Pi
    - GND → GND de la Raspberry Pi
    - DATA → GPIO 4 (con resistencia pull-up de 10kΩ)

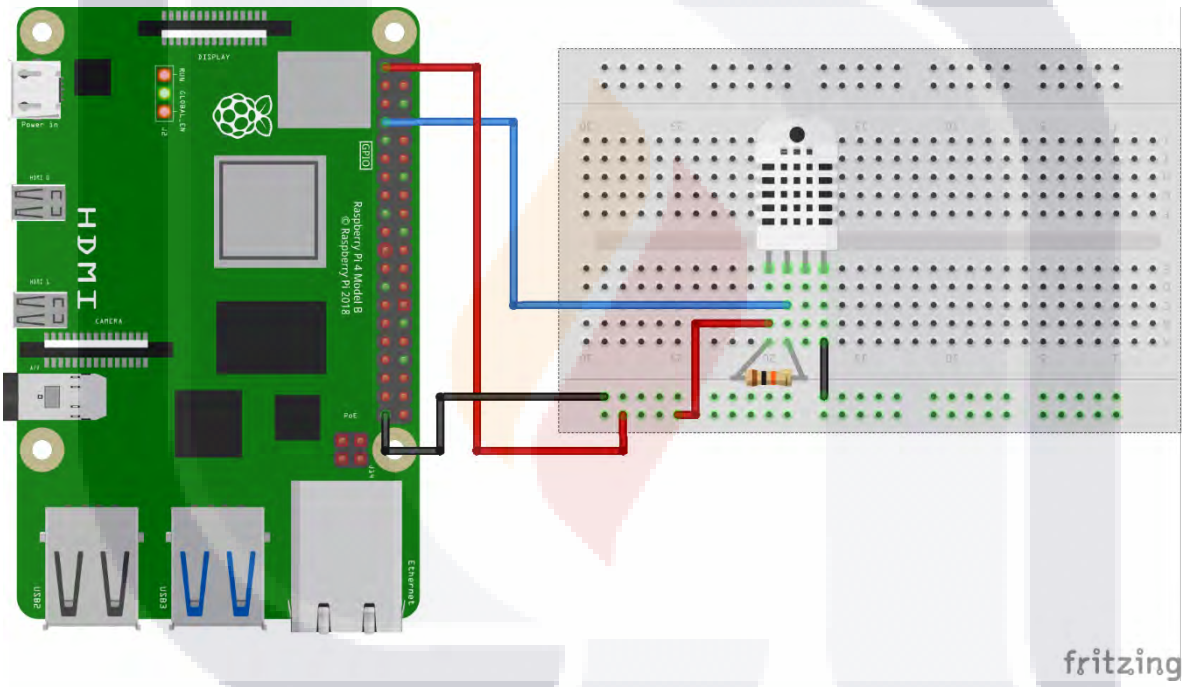


Diagrama 6. Conexión del sensor DHT22 con la Raspberry Pi.

Conexión del DS18B20 (Temperatura del suelo)

- Alimentación: 3.3V o 5V
- Comunicación: 1-Wire
- Pines de conexión:

- VCC → 3.3V de la Raspberry Pi
- GND → GND de la Raspberry Pi
- DATA → GPIO 4 (compartido con el DHT22, requiere resistencia pull-up de 4.7kΩ)

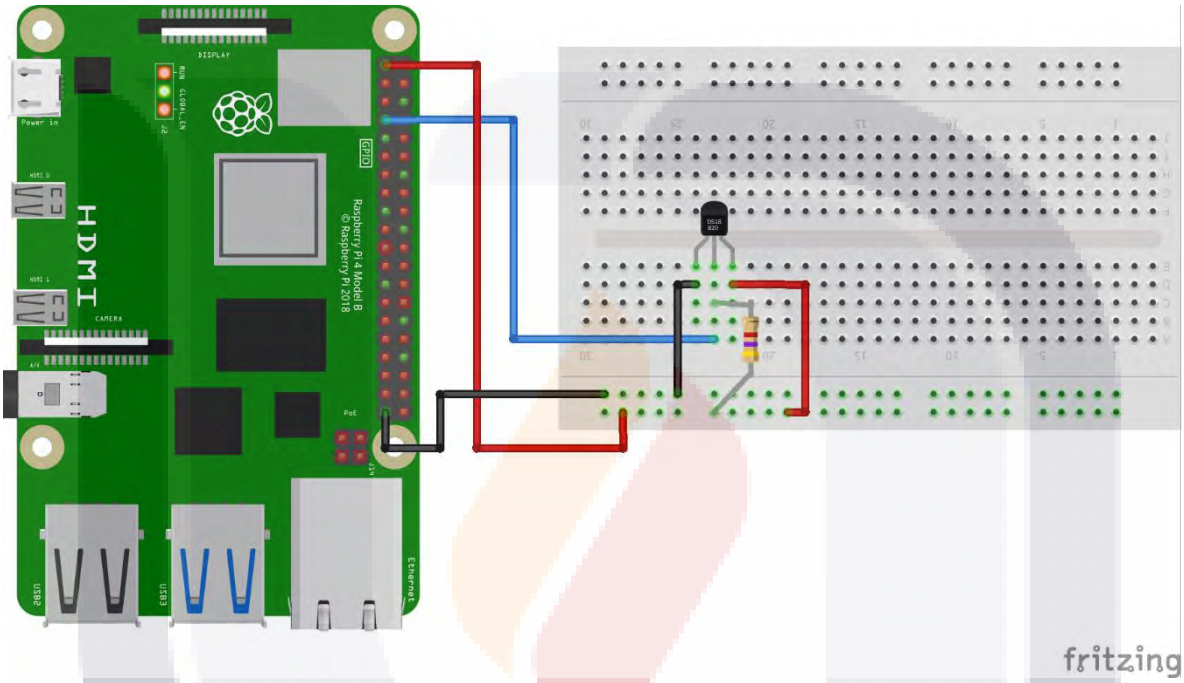


Diagrama 7. Conexión del sensor DS18B20 con la Raspberry Pi.

Conexión del BH1750 (Intensidad de luz ambiental) en el Bus I²C 3

- Alimentación: 3.3V o 5V
- Comunicación: I²C
- Pines de conexión:
  - VCC → 3.3V de la Raspberry Pi
  - GND → GND de la Raspberry Pi

- SDA → GPIO 23 (SDA bus 3)
- SCL → GPIO 24 (SCL bus 3)

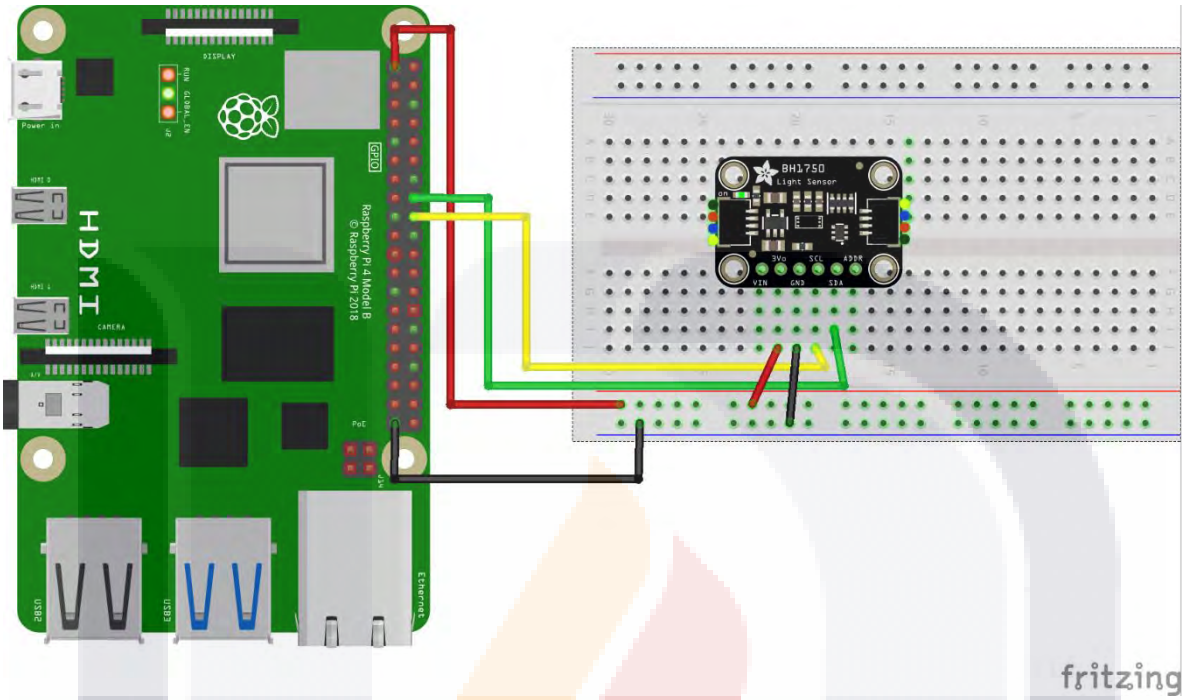


Diagrama 8. Conexión del sensor BH1750 con la Raspberry Pi.

Conexión del SCD41 (Sensor de CO_2) en el Bus I²C 4

- Alimentación: 3.3V o 5V
- Comunicación: I²C
- Pines de conexión:
  - VCC → 3.3V de la Raspberry Pi
  - GND → GND de la Raspberry Pi
  - SDA → GPIO 27 (SDA bus 4)

- SCL → GPIO 22 (SCL bus 4)

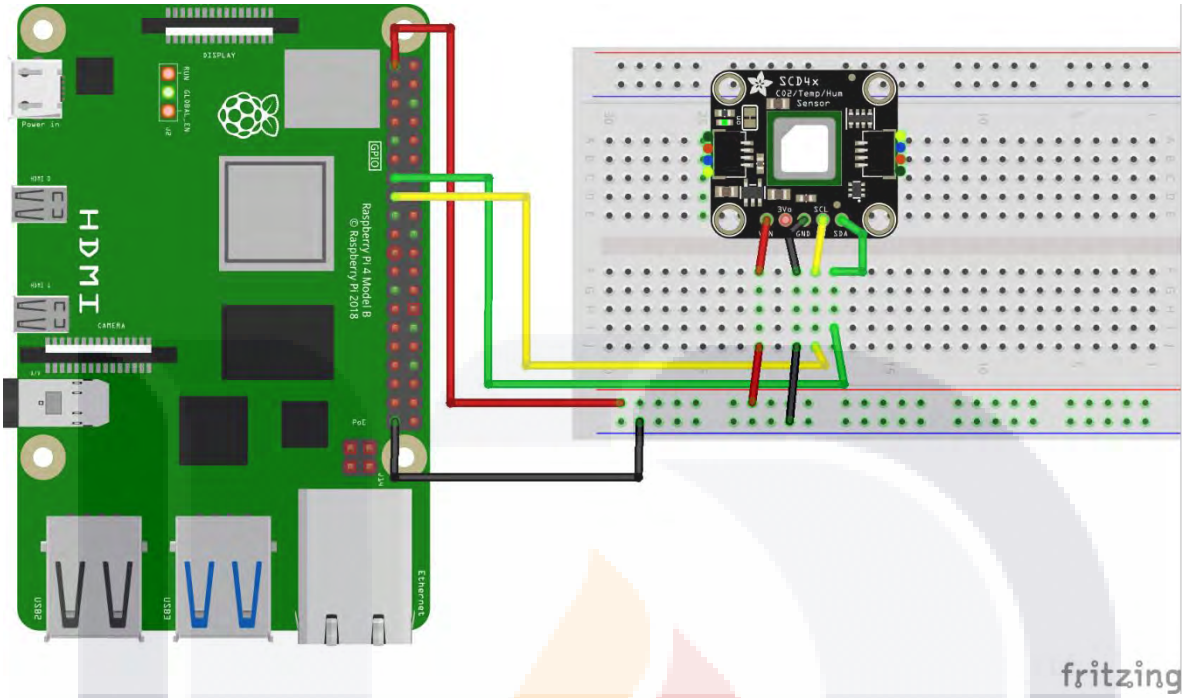


Diagrama 9. Conexión del sensor SCD41 con la Raspberry Pi.

Conexión del LM393 (Sensor de humedad del suelo con ADC ADS1115) en el Bus I²C 3

Dado que la Raspberry Pi no cuenta con entradas analógicas, se utiliza el convertidor ADC ADS1115 para leer la salida analógica del sensor LM393.

- Alimentación: 3.3V o 5V
- Conversión ADC: ADS1115 (16 bits)
- Conexión del LM393 al ADS1115:
  - VCC → 3.3V de la Raspberry Pi
  - GND → GND de la Raspberry Pi
  - A0 (Salida Analógica) → A0 del ADS1115

- Conexión del ADS1115 a la Raspberry Pi:
    - VCC → 3.3V de la Raspberry Pi
    - GND → GND de la Raspberry Pi
    - SDA → GPIO 23 (SDA bus 3)
    - SCL → GPIO 24 (SCL bus 3)

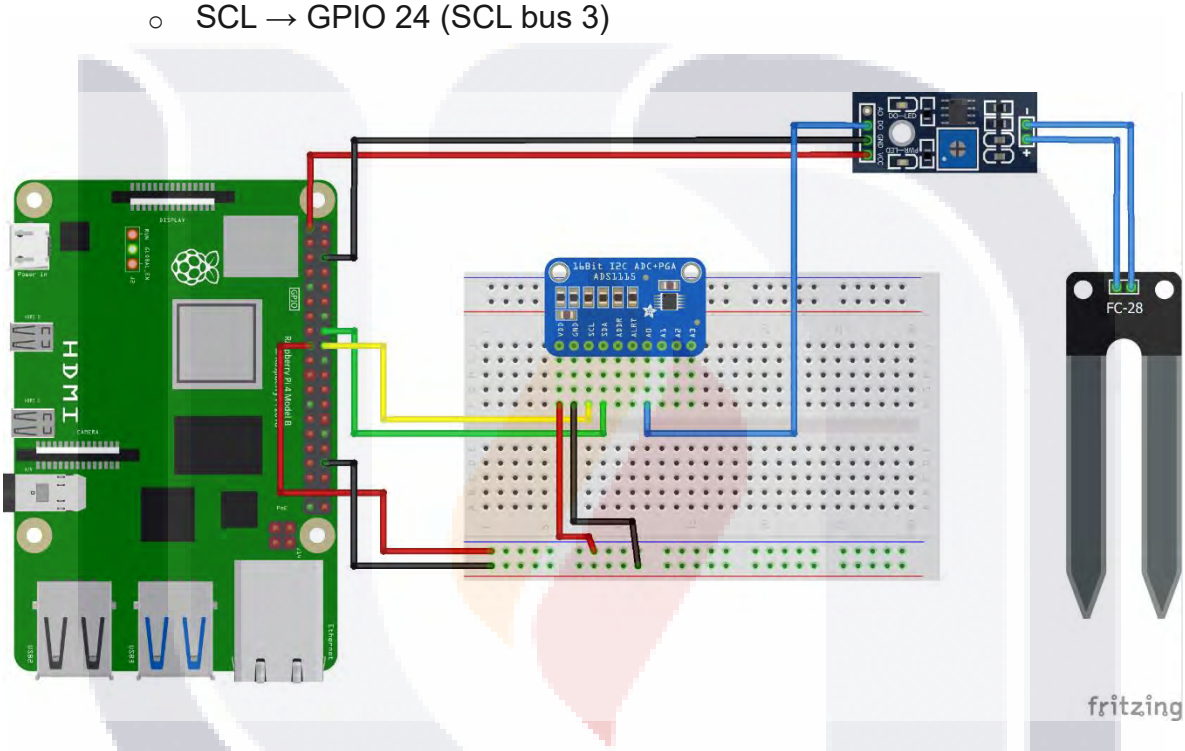


Diagrama 10. Conexión del LM393 con el ADS1115 y la Raspberry Pi.

Esta configuración de hardware permite que el sistema IoT maneje múltiples sensores sin conflictos de dirección en el bus I²C, garantizando una comunicación eficiente y estable con la Raspberry Pi [66].

Esquema de Conexión de la Bomba de Agua

La bomba de agua es el único actuador del sistema de riego inteligente basado en IoT y es controlada mediante un módulo relé conectado a la Raspberry Pi. La

activación de la bomba se basa en los valores obtenidos del sensor de humedad del suelo (LM393), permitiendo una gestión eficiente del riego.

Componentes Utilizados

- Bomba de agua.
- Módulo relé de 1 canal.
- Fuente de alimentación de la bomba.
- Raspberry Pi 4 Model B.

Conexión del Módulo Relé con la Raspberry Pi

El módulo relé actúa como un interruptor controlado digitalmente por la Raspberry Pi para encender o apagar la bomba de agua. La conexión se realiza de la siguiente manera:

- VCC → 5V de la Raspberry Pi
- GND → GND de la Raspberry Pi
- IN → GPIO 17 (puede cambiarse según necesidad)

Cuando la Raspberry Pi envía un nivel lógico bajo al pin IN, el relé se activa y permite el paso de corriente hacia la bomba de agua.

Conexión de la Bomba de Agua al Relé

La bomba de agua opera con una fuente de alimentación externa (3v – 6V), y su circuito de control se establece a través del relé.

1. Conexión en el lado de control del relé:
    - Un terminal de la fuente de alimentación se conecta a COM (común) del relé.

- El otro terminal de la fuente de alimentación se conecta directamente a la bomba de agua.
- El pin NO (normalmente abierto) del relé se conecta al otro terminal de la bomba.

2. Flujo de operación:

- Cuando el relé está inactivo, el circuito de la bomba está abierto y no fluye corriente.
- Cuando el relé es activado por la Raspberry Pi, el circuito se cierra y la bomba comienza a funcionar.

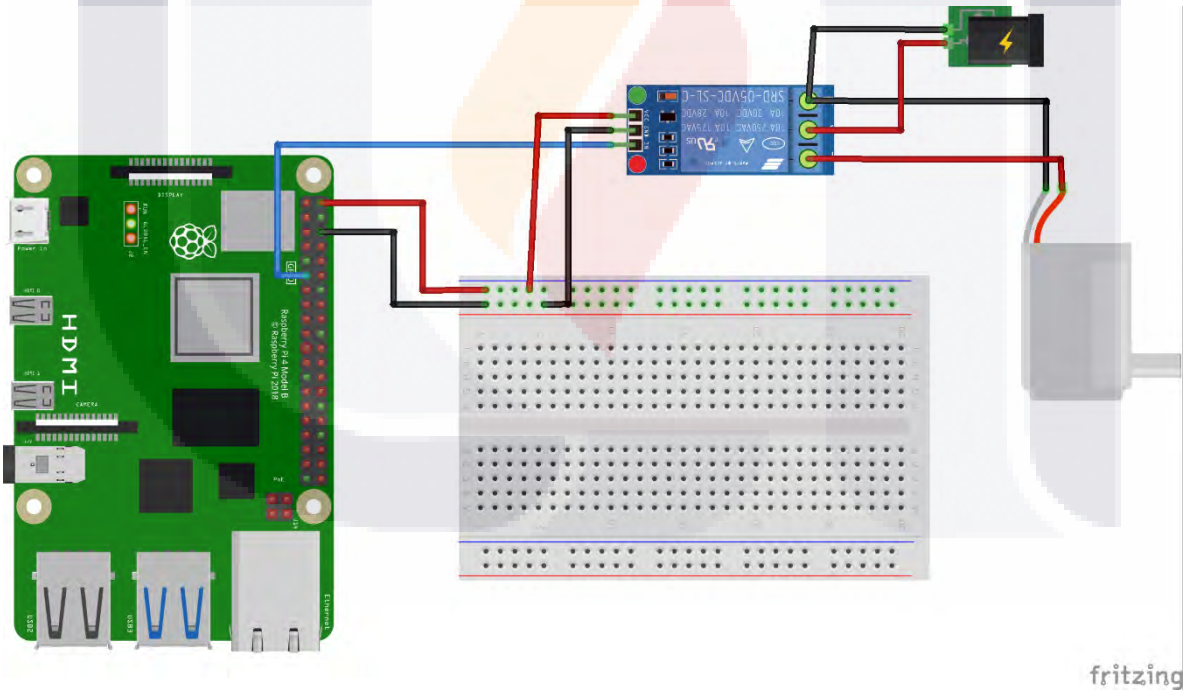


Diagrama 11. Conexión eléctrica de la bomba de agua con el relé y la Raspberry Pi.

Esta configuración garantiza un control preciso del riego, asegurando que el agua se distribuya solo cuando el nivel de humedad del suelo lo requiera [44].

A.4. Lectura de sensores

Código de Captura de Datos de Sensores

El sistema de riego inteligente recopila información de múltiples sensores ambientales y del suelo, utilizando scripts individuales en Python para cada sensor. Estos scripts son gestionados desde el script principal **`mqtt_client.py`**, el cual importa y ejecuta las funciones de cada sensor.

Estructura del Código

El código se organiza de la siguiente manera:

```
/sistema_riego_iot/
|— .venv/                    # Entorno virtual de Python
|— mqtt_client.py            # Cliente MQTT principal
|— sensores/
|   |— dht22.py              # Script para el sensor DHT22
|   |— ds18b20.py            # Script para el sensor DS18B20
|   |— bh1750.py             # Script para el sensor BH1750
|   |— scd41.py              # Script para el sensor SCD41
|   |— lm393_ads1115.py      # Script para el sensor de humedad del suelo LM393 con ADS1115
```

Código 22. Estructura de carpetas de scripts de sensores.

Cada script de sensor define una función para leer los datos y retornarlos al cliente MQTT para su publicación.

Código de Lectura del Sensor DHT22 (Temperatura y Humedad)

```python
import adafruit_dht

class DHT22:
    def __init__(self, pin):
        # Inicializa el sensor DHT22 en el pin especificado usando `board`
        self.sensor = adafruit_dht.DHT22(pin)

    def read_data(self):
        """Obtiene la temperatura y la humedad desde el sensor DHT22."""
        try:
            # Lee la temperatura y la humedad desde el sensor
            temperature = self.sensor.temperature
            humidity = self.sensor.humidity

            # Verifica si los datos son válidos
            if humidity is not None and temperature is not None:
                return {"temperatura": temperature, "humedad": humidity}
            else:
                print("Error al leer el sensor DHT22.")
                return None
        except RuntimeError as error:
            # Ocurre un RuntimeError ocasionalmente cuando la lectura falla
            print(f"Error al leer el sensor DHT22: {error}")
            return None
```

Código 23. Código de Lectura del Sensor DHT22.

Código de Lectura del Sensor DS18B20 (Temperatura del Suelo)

```python
from w1thermsensor import W1ThermSensor

class DS18B20:
    def __init__(self):
        # Inicializa el sensor usando la libreria w1thermsensor
        self.sensor = W1ThermSensor()

    def read_temperature(self):
        """Obtiene la temperatura en grados Celsius del sensor DS18B20."""
        try:
            temp_celsius = self.sensor.get_temperature()  # Obtiene la temperatura en Celsius
            return temp_celsius
        except Exception as e:
            print(f"Error al leer el sensor DS18B20: {e}")
            return None
```

Código 24. Código de Lectura del Sensor DS18B20.

Código de Lectura del Sensor BH1750 (Intensidad de Luz)

```python
from smbus2 import SMBus
import time

class BH1750:
    """Clase para manejar el sensor de luz BH1750 usando SMBus."""

    # Direcciones posibles del sensor
    ADDRESS_LOW = 0x23  # Dirección por defecto
    ADDRESS_HIGH = 0x5C # Dirección alternativa

    # Modos de operación
    CONTINUOUS_HIGH_RES_MODE = 0x10  # 1 lx, 120ms
    CONTINUOUS_HIGH_RES_MODE_2 = 0x11 # 0.5 lx, 120ms
    CONTINUOUS_LOW_RES_MODE = 0x13    # 4 lx, 16ms
    ONE_TIME_HIGH_RES_MODE = 0x20     # 1 lx, 120ms, luego apaga
    ONE_TIME_LOW_RES_MODE = 0x23      # 4 lx, 16ms, luego apaga

    def __init__(self, bus_number=1, address=ADDRESS_LOW, mode=CONTINUOUS_HIGH_RES_MODE):
        """
        Inicializa el sensor BH1750.

        :param bus_number: Número del bus I2C (por defecto 1).
        :param address: Dirección I2C del sensor (0x23 o 0x5C).
        :param mode: Modo de medición del sensor.
        """
        self.bus_number = bus_number
        self.address = address
        self.mode = mode
        self.bus = SMBus(self.bus_number)  # Inicializar el bus I2C

        # Encender el sensor al iniciar
        self.power_on()
```

Código 25. Código de Lectura del Sensor BH1750 (parte 1).

```
    def power_on(self):
        """Enciende el sensor BH1750."""
        self.bus.write_byte(self.address, 0x01)

    def reset(self):
        """Reinicia el sensor BH1750."""
        self.bus.write_byte(self.address, 0x07)

    def set_mode(self, mode):
        """Establece el modo de medición."""
        self.mode = mode
        self.bus.write_byte(self.address, self.mode)

    def read_light(self):
        """Lee el valor de luz en lux."""
        try:
            # Enviar comando de medición
            self.bus.write_byte(self.address, self.mode)
            time.sleep(0.2)  # Esperar 120ms para completar la medición

            # Leer 2 bytes de datos
            data = self.bus.read_i2c_block_data(self.address, self.mode, 2)
            light_level = (data[0] << 8) | data[1]  # Convertir a 16 bits
            return light_level / 1.2  # Ajuste según la documentación

        except Exception as e:
            print(f"Error al leer el BH1750: {e}")
            return None
```

Código 26. Código de Lectura del Sensor BH1750 (parte 2).

```
    def close(self):
        """Cierra el bus I2C cuando se deja de usar el sensor."""
        self.bus.close()

# Uso del sensor en I2C-5
if __name__ == "__main__":
    sensor = BH1750(bus_number=4, address=BH1750.ADDRESS_LOW)

    try:
        while True:
            luz = sensor.read_light()
            if luz is not None:
                print(f"Luz: {luz:.2f} lx")
            time.sleep(1)
    except KeyboardInterrupt:
        print("Apagando sensor...")
        sensor.close()
```

Código 27. Código de Lectura del Sensor BH1750 (parte 3).

Código de Lectura del Sensor SCD41 (CO_2)

```python
from smbus2 import SMBus, i2c_msg
import time

class SCD41:
    """Clase para manejar el sensor SCD41 en Raspberry Pi con smbus2."""

    ADDRESS = 0x62  # Dirección I2C del SCD41

    # Comandos Sensirion
    START_PERIODIC_MEASUREMENT = [0x21, 0xB1]
    STOP_MEASUREMENT = [0x3F, 0x86]
    READ_MEASUREMENT = [0xEC, 0x05]

    def __init__(self, bus_number=1):
        """Inicializa el sensor SCD41 en el bus I2C especificado."""
        self.bus_number = bus_number
        self.bus = SMBus(self.bus_number)

        # Iniciar medición periódica
        self.send_command(self.START_PERIODIC_MEASUREMENT)
        time.sleep(1)  # Esperar a que el sensor inicie correctamente

    def send_command(self, command):
        """Envía un comando I2C al sensor."""
        self.bus.write_i2c_block_data(self.ADDRESS, command[0], command[1:])
```

Código 28. Código de Lectura del Sensor SCD41 (parte 1).

```python
    def read_measurement(self):
        """Lee la medición de CO2, temperatura y humedad."""
        try:
            # Enviar comando para leer datos
            self.send_command(self.READ_MEASUREMENT)
            time.sleep(0.2)  # Esperar respuesta del sensor

            # Leer 9 bytes de datos
            msg = i2c_msg.read(self.ADDRESS, 9)
            self.bus.i2c_rdwr(msg)

            # Convertir los datos
            raw = list(msg)
            co2 = (raw[0] << 8) | raw[1]
            temp_raw = (raw[3] << 8) | raw[4]
            humidity_raw = (raw[6] << 8) | raw[7]

            # Convertir valores a unidades reales
            temperature = -45 + (175 * (temp_raw / 65535.0))
            humidity = 100 * (humidity_raw / 65535.0)

            return {"CO2": co2, "Temperature": temperature, "Humidity": humidity}

        except Exception as e:
            print(f"Error al leer el SCD41: {e}")
            return None

    def stop_measurement(self):
        """Detiene la medición del sensor SCD41."""
        self.send_command(self.STOP_MEASUREMENT)
        time.sleep(1)
```

Código 29. Código de Lectura del Sensor SCD41 (parte 2).

```
    def close(self):
        """Cierra el bus I2C."""
        self.bus.close()

# Uso del sensor en I2C-4
if __name__ == "__main__":
    sensor = SCD41(bus_number=5)

    try:
        while True:
            data = sensor.read_measurement()
            if data:
                print(f"CO2: {data['CO2']} ppm | Temp: {data['Temperature']:.2f} °C | Humedad: {data['Humidity']:.2f}
%")
            time.sleep(5)
    except KeyboardInterrupt:
        print("Deteniendo sensor...")
        sensor.stop_measurement()
        sensor.close()
```

Código 30. Código de Lectura del Sensor SCD41 (parte 3).

Código de Lectura del Sensor de Humedad del Suelo LM393 con ADS1115

```
import time
import board
import busio
import adafruit_ads1x15.ads1115 as ADS
from adafruit_ads1x15.analog_in import AnalogIn

class SoilMoistureSensor:
    """Class to read soil moisture levels using ADS1115 and an analog sensor."""

    def __init__(self, scl=board.SCL, sda=board.SDA, channel=ADS.P0, dry_value=22000, wet_value=12000):
        """
        Initializes communication with the ADS1115 ADC and sets the reading channel.

        :param scl: I2C clock pin (default SCL from Raspberry Pi).
        :param sda: I2C data pin (default SDA from Raspberry Pi).
        :param channel: ADS1115 channel where the sensor is connected.
        :param dry_value: ADC value for dry soil (adjust based on calibration).
        :param wet_value: ADC value for wet soil (adjust based on calibration).
        """
        self.i2c = busio.I2C(scl, sda)  # Initialize I2C bus
        self.ads = ADS.ADS1115(self.i2c)  # Initialize ADS1115 ADC
        self.channel = AnalogIn(self.ads, channel)  # Set reading channel
        self.dry_value = dry_value
        self.wet_value = wet_value

    def read_adc(self):
        """Reads the digital ADC value (16-bit)."""
        return self.channel.value

    def read_voltage(self):
        """Reads the voltage converted by the ADC."""
        return self.channel.voltage
```

Código 31. Lectura del Sensor de Humedad del Suelo LM393 con ADS1115 (parte 1).

```
    def calculate_moisture(self):
        """Converts the ADC value into a moisture percentage."""
        adc_value = self.read_adc()
        moisture = 100 * (self.dry_value - adc_value) / (self.dry_value - self.wet_value)
        return max(0, min(100, moisture))  # Ensure the value is between 0% and 100%

    def get_data(self):
        """Returns a dictionary with all sensor readings."""
        return {
            "ADC": self.read_adc(),
            "Voltage": self.read_voltage(),
            "Moisture": self.calculate_moisture()
        }

# --- Execution Test ---
if __name__ == "__main__":
    soil_sensor = SoilMoistureSensor()

    try:
        while True:
            data = soil_sensor.get_data()
            print(f"ADC: {data['ADC']} | Voltage: {data['Voltage']:.2f}V | Moisture: {data['Moisture']:.2f}%")
            time.sleep(1)
    except KeyboardInterrupt:
        print("Stopping soil moisture sensor reading...")
```

Código 32. Lectura del Sensor de Humedad del Suelo LM393 con ADS1115 (parte 2).

Con esta arquitectura modular, cada sensor tiene su propio script, lo que facilita la escalabilidad y mantenimiento del sistema IoT. Scripts en Python para la lectura de los sensores y procesamiento de los datos obtenidos.

A.5 Control de actuadores

Código de Control de la Bomba de Agua

El control de la bomba de agua en el sistema de riego inteligente se realiza a través de un módulo relé conectado a la Raspberry Pi. La activación y desactivación de la bomba dependen de los valores obtenidos por el sensor de humedad del suelo LM393 con ADS1115.

Estructura del Código

El código sigue la misma estructura modular utilizada en los sensores y se encuentra organizado de la siguiente manera:

```
/sistema_riego_iot/
|— .venv/                       # Entorno virtual de Python
|— mqtt_client.py               # Cliente MQTT principal
|— sensores/
|   |— dht22.py                 # Script para el sensor DHT22
|   |— ds18b20.py               # Script para el sensor DS18B20
|   |— bh1750.py                # Script para el sensor BH1750
|   |— scd41.py                 # Script para el sensor SCD41
|   |— lm393_ads1115.py         # Script para el sensor de humedad del suelo LM393 con ADS1115
|— control/
|   |— relay_bomba.py           # Script para accionar la bomba de agua
```

Código 33. Estructura de carpetas de scripts de actuadores.

Código de Control de la Bomba de Agua (relay_bomba.py)

```python
import RPi.GPIO as GPIO

class Relay:
    def __init__(self, pin):
        self.pin = pin
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.pin, GPIO.OUT)
        GPIO.output(self.pin, GPIO.HIGH)  # Relé apagado por defecto

    def turn_on(self):
        GPIO.output(self.pin, GPIO.LOW)  # Activar relé
        print("Bomba encendida")

    def turn_off(self):
        GPIO.output(self.pin, GPIO.HIGH)  # Desactivar relé
        print("Bomba apagada")

    def cleanup(self):
        GPIO.cleanup()
```

Código 34. Código de Control de la Bomba de Agua.

Este código garantiza que la bomba de agua se active solo cuando sea necesario, optimizando el uso del recurso hídrico en el sistema de riego inteligente basado en IoT.

A.6 Implementación del cliente MQTT

Código de Comunicación MQTT

La comunicación en el sistema de riego inteligente se realiza utilizando el protocolo MQTT, un estándar ampliamente utilizado en IoT debido a su eficiencia y bajo consumo de ancho de banda. Este protocolo permite que la Raspberry Pi envíe datos de los sensores y reciba comandos para la activación y desactivación de la bomba de agua a través del broker de HiveMQ [47].

Estructura de Comunicación MQTT

El sistema sigue una arquitectura publicador-suscriptor, donde la Raspberry Pi actúa como:

- Publicador: Enviando datos de sensores a tópicos específicos en el broker.
- Suscriptor: Recibiendo comandos para activar o desactivar la bomba de agua [47].

Los tópicos utilizados en este sistema son:

- **`iot/sensores`** → Publica los valores de temperatura, humedad, luz, CO_2 y humedad del suelo.
- **`iot/control`** → Recibe comandos (ON u OFF) para el control de la bomba de agua.

Instalación de la Biblioteca MQTT

Para habilitar la comunicación MQTT en Python, es necesario instalar la biblioteca **`paho-mqtt`** si no está instalada previamente:

```
pip install paho-mqtt
```

Código de Cliente MQTT en `mqtt_client.py`

El siguiente código implementa un cliente MQTT en la Raspberry Pi para gestionar la comunicación con HiveMQ:

```python
import os
import time
import json
import board
from dotenv import load_dotenv
import paho.mqtt.client as paho
from paho import mqtt
from sensors.dht22 import DHT22
from sensors.ds18b20 import DS18B20
from sensors.bh1750 import BH1750
from sensors.scd41 import SCD41
from sensors.ads1115 import SoilMoistureSensor
from control.relay import Relay


# Cargar las variables de entorno desde el archivo .env
load_dotenv()

# Configuración del broker MQTT en HiveMQ Cloud
MQTT_BROKER = os.getenv('BROKER_ADDRESS')
MQTT_PORT = int(os.getenv('BROKER_PORT'))
MQTT_TOPIC = os.getenv('MQTT_TOPIC')

# Credenciales del cliente (cambiar por tu usuario y contraseña de HiveMQ)
MQTT_USERNAME = os.getenv('BROKER_USERNAME')
MQTT_PASSWORD = os.getenv('BROKER_PASSWORD')

# Configuración de buses I2C
BH1750_I2C_BUS = int(os.getenv('BH1750_I2C_BUS'))
SCD41_I2C_BUS = int(os.getenv('SCD41_I2C_BUS'))

# Configuración de pines de sensores
DHT22_PIN = os.getenv('DHT22_PIN')

# Configuración de pines de control
RELAY_PIN = int(os.getenv('RELAY_PIN'))

# Intervalo de tiempo entre lecturas (en segundos)
READ_INTERVAL = int(os.getenv('READ_INTERVAL', 5))

# Inicializar los sensores
ds18b20_sensor = DS18B20()
dht22_sensor = DHT22(pin=getattr(board, DHT22_PIN))
bh1750_sensor = BH1750(bus_number=BH1750_I2C_BUS)
scd41_sensor = SCD41(bus_number=SCD41_I2C_BUS)
soilMoisture_sensor = SoilMoistureSensor()
relay = Relay(RELAY_PIN)
```

Código 35. Cliente MQTT en mqtt_client.py (parte 1).

```python
# Función para publicar datos en el broker
def publish_data(client):
    try:
        # Lee la temperatura del DS18B20
        ds18b20_data = ds18b20_sensor.read_temperature()

        # Lee la temperatura y humedad del DHT22
        dht22_data = dht22_sensor.read_data()

        # Lee los niveles de luz de BH1750
        bh1750_data = bh1750_sensor.read_light()

        # Lee los niveles de CO2 de SCD41
        scd41_data = scd41_sensor.read_measurement()

        # Lee los niveles de humedad de suelo de ADS1115
        soilMoistureSensor_data = soilMoisture_sensor.get_data()

        # Preparar los datos en un formato JSON
        data = [
            {
                "name": "Soil Temperature",
                "value": ds18b20_data if ds18b20_data else None,
            },
            {
                "name": "Air Temperature",
                "value": dht22_data["temperatura"] if dht22_data else None,
            },
            {
                "name": "Air Humidity",
                "value": dht22_data["humedad"] if dht22_data else None,
            },
            {
                "name": "Light",
                "value": bh1750_data if bh1750_data else None,
            },
            {
                "name": "CO2",
                "value": scd41_data["CO2"] if scd41_data else None,
            },
            {
                "name": "Soil Moisture",
                "value": soilMoistureSensor_data["Moisture"] if soilMoistureSensor_data else None,
            },
        ]

        # Convierte los datos a JSON
        message = json.dumps(data)

        # Publica los datos en el topic
        client.publish(MQTT_TOPIC, message)
        print(f"Publicado: {message}")
    except Exception as e:
        print(f"Error al leer o enviar la lectura: {e}")
```

Código 36. Cliente MQTT en mqtt_client.py (parte 2).

```python
# Callback cuando se recibe un mensaje MQTT
def on_message(client, userdata, message):
    try:
        payload = json.loads(message.payload.decode("utf-8"))
        command = payload.get("command", "").upper()
        device = payload.get("device", "")

        print(f"Mensaje recibido: {payload}")

        if device == "bomba":
            if command == "ON":
                relay.turn_on()
            elif command == "OFF":
                relay.turn_off()
            else:
                print("Comando desconocido")
        else:
            print("Dispositivo no reconocido")

    except json.JSONDecodeError:
        print("Error: Mensaje no es un JSON válido")

# Callback que se ejecuta cuando el cliente se conecta al broker
def on_connect(client, userdata, flags, rc, properties):
    if rc == 0:
        print("Conectado exitosamente al broker MQTT")
        client.subscribe("iot/control")
    else:
        print(f"Conexión fallida con código {rc}")

# Callback que se ejecuta en caso de error
def on_log(client, userdata, level, buf):
    print(f"Log: {buf}")
```

Código 37. Cliente MQTT en mqtt_client.py (parte 3).

```python
# Configuración del cliente MQTT
client = paho.Client(paho.CallbackAPIVersion.VERSION2)

# Autenticación con usuario y contraseña
client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)

# Configuración de TLS/SSL
client.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)

# Configurar callbacks
client.on_connect = on_connect
client.on_log = on_log  # Para ver detalles de la conexión
client.on_message = on_message

# Conexión al broker
client.connect(MQTT_BROKER, MQTT_PORT, 60)

# Bucle principal
client.loop_start()  # Iniciar el loop de MQTT en segundo plano
try:
    while True:
        # data = get_sensor_data()
        # if data:
        #     client.publish(MQTT_TOPIC, json.dumps(data))
        #     print(f"Temperatura enviada: {data.temperature:.2f} °C")
        publish_data(client)
        time.sleep(READ_INTERVAL)  # Esperar 10 segundos antes de la siguiente lectura
except KeyboardInterrupt:
    print("Finalizando...")
finally:
    client.loop_stop()
    scd41_sensor.stop_measurement()
    scd41_sensor.close()
    bh1750_sensor.close()
    relay.cleanup()
    client.disconnect()
```

Código 38. Cliente MQTT en mqtt_client.py (parte 4).

Explicación del Código

1. Conexión al broker MQTT: Se establece la conexión con HiveMQ y se suscribe al tópico **`iot/control`** para recibir comandos.

2. Recepción de mensajes: La función **`on_message`** procesa los mensajes recibidos y activa o desactiva la bomba según el comando.

3. Publicación de datos: Cada 10 segundos, la Raspberry Pi recopila los valores de los sensores y los envía al tópico **`iot/sensores`**.

Este código asegura una comunicación eficiente entre la Raspberry Pi y el broker MQTT de HiveMQ, permitiendo la monitorización en tiempo real del sistema de riego inteligente.

Publicación de Datos de Sensores

La Raspberry Pi adquiere información de los sensores y la envía al broker MQTT en un formato estructurado. La publicación de estos datos se realiza en el tópico:

`iot/sensores`

Cada sensor genera datos que se publican periódicamente en el broker para su monitoreo en tiempo real. Ejemplo de publicación de datos:

```
import os
import time
import json
import board
from dotenv import load_dotenv
import paho.mqtt.client as paho
from paho import mqtt
from sensors.dht22 import DHT22
from sensors.ds18b20 import DS18B20
from sensors.bh1750 import BH1750
from sensors.scd41 import SCD41
from sensors.ads1115 import SoilMoistureSensor
from control.relay import Relay

ds18b20_sensor = DS18B20()
dht22_sensor = DHT22(pin=getattr(board, DHT22_PIN))
bh1750_sensor = BH1750(bus_number=BH1750_I2C_BUS)
scd41_sensor = SCD41(bus_number=SCD41_I2C_BUS)
soilMoisture_sensor = SoilMoistureSensor()
relay = Relay(RELAY_PIN)

try:
    while True:
        publish_data(client)
        time.sleep(READ_INTERVAL)  # Esperar 10 segundos antes de la siguiente lectura
except KeyboardInterrupt:
    print("Finalizando...")
```

Código 39. Ejemplo de Publicación de Datos de Sensores.

Recepción de Comandos para la Bomba de Agua

La Raspberry Pi se suscribe al tópico:

`iot/control`

En este tópico se reciben comandos para la activación (ON) o desactivación (OFF) de la bomba de agua. El siguiente código muestra cómo se suscribe y gestiona los mensajes recibidos:

```
# Callback cuando se recibe un mensaje MQTT
def on_message(client, userdata, message):
    try:
        payload = json.loads(message.payload.decode("utf-8"))
        command = payload.get("command", "").upper()
        device = payload.get("device", "")

        print(f"Mensaje recibido: {payload}")

        if device == "bomba":
            if command == "ON":
                relay.turn_on()
            elif command == "OFF":
                relay.turn_off()
            else:
                print("Comando desconocido")
        else:
            print("Dispositivo no reconocido")

    except json.JSONDecodeError:
        print("Error: Mensaje no es un JSON válido")

client.on_message = on_message
```

Código 40. Recepción de Comandos para la Bomba de Agua.

Flujo de Comunicación

1. Adquisición de Datos: Los sensores miden temperatura, humedad, luz, CO_2 y humedad del suelo.
2. Publicación de Datos: La Raspberry Pi publica estos valores en **`iot/sensores`**.
3. Recepción de Comandos: Si un usuario o sistema externo publica ON u OFF en **`iot/control`**, la Raspberry Pi procesa la orden y activa/desactiva la bomba.

Anexo B. Implementación del backend y frontend web

B.1 Configuración del backend

Instalación y Configuración de Celery

Para usar Celery, primero es necesario instalar la librería y configurar el backend de tareas y el scheduler (programador de tareas). En este caso, utilizaremos Redis como el broker de Celery, que es el intermediario encargado de gestionar las colas de tareas.

Instalación de Celery y Redis:

Primero, instalamos Celery y Celery Beat (para programación de tareas periódicas) junto con Redis como backend:

```
pip install celery
```

```
pip install celery[redis]
```

Configuración en `settings.py`:

En el archivo settings.py de Django, se configura el broker (Redis) y el backend de Celery para que se comuniquen de manera eficiente.

```
# settings.py

CELERY_BROKER_URL = 'redis://localhost:6379/0'  # Dirección de Redis
CELERY_RESULT_BACKEND = 'redis://localhost:6379/0'  # Backend para almacenar
resultados
CELERY_ACCEPT_CONTENT = ['json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_TIMEZONE = 'UTC'
```

Código 41. Configuración en settings.py.

Esto configura Redis como el broker para manejar las tareas de Celery y también establece cómo se serializan los datos entre los componentes (en este caso, JSON) [34].

Configuración del Cliente MQTT en Django

El backend se conecta al bróker HiveMQ utilizando un cliente MQTT. Para ello, se puede utilizar la librería **`paho-mqtt`**, que es una de las bibliotecas más utilizadas para trabajar con MQTT en Python.

A continuación, se muestra cómo configurar el cliente MQTT en el backend de Django para conectarse a HiveMQ, suscribirse a un tema y recibir los mensajes.

```python
import paho.mqtt.client as mqtt
from django.conf import settings

# Callback cuando se conecta al bróker
def on_connect(client, userdata, flags, rc):
    print(f"Conectado al bróker con código {rc}")
    client.subscribe("sensores/#")  # Suscripción a todos los temas de sensores

# Callback cuando se recibe un mensaje
def on_message(client, userdata, msg):
    print(f"Mensaje recibido: {msg.payload}")
    # Aquí procesamos los datos recibidos
    process_sensor_data(msg.payload)

# Función para procesar los datos de los sensores
def process_sensor_data(data):
    # Parsear y almacenar los datos en la base de datos
    # Aquí se puede convertir el mensaje recibido en un formato útil
    sensor_data = data.decode("utf-8")
    # Ejemplo de cómo guardar los datos en la base de datos:
    SensorData.objects.create(value=sensor_data)

# Configuración del cliente MQTT
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# Conexión al bróker HiveMQ
client.connect(settings.MQTT_BROKER_HOST, settings.MQTT_BROKER_PORT, 60)

# Bucle para mantener la conexión activa
client.loop_start()
```

Código 42. Código de Conexión MQTT en Django.

Explicación del código:

1. Conexión al bróker: El cliente MQTT se conecta al bróker HiveMQ utilizando las credenciales y el host configurado en los ajustes de Django.

2. Suscripción a los temas: Una vez conectado, el cliente se suscribe al tema **`iot/sensores`**, lo que le permite recibir todos los mensajes publicados en los temas relacionados con los sensores.
3. Recepción de mensajes: Cuando el cliente recibe un mensaje, el callback **`on_message`** es ejecutado. Los datos del mensaje (que generalmente estarán en formato JSON o texto) se procesan y se almacenan en la base de datos.
4. Procesamiento de los datos: Los datos del sensor se procesan en la función **`process_sensor_data`**, donde los mensajes recibidos se convierten a un formato adecuado para ser almacenados en la base de datos [64].

Manejo de Conexiones y Reconexión Automática

Una de las características más importantes del protocolo MQTT es su capacidad de manejar conexiones inestables o intermitentes, lo que es especialmente útil en entornos como un cultivo, donde las conexiones pueden ser inestables. **`paho-mqtt`** soporta la reconexión automática en caso de que el cliente pierda la conexión con el bróker.

```
def on_disconnect(client, userdata, rc):
    print(f"Desconectado con código {rc}")
    if rc != 0:
        print("Reconectando...")
        client.reconnect()  # Intentar reconectar en caso de desconexión
inesperada

client.on_disconnect = on_disconnect
```

Código 43. Manejo de Reconexión Automática.

Explicación:

- En caso de desconexión inesperada, el cliente intentará reconectarse automáticamente, asegurando que el backend reciba los datos de los sensores de manera continua [64].

Manejo de Errores en la Comunicación MQTT y WebSocket

Es importante implementar un manejo robusto de errores para asegurar la estabilidad del sistema, ya que tanto en la comunicación MQTT (para recibir datos de los sensores) como en los WebSockets (para la comunicación en tiempo real con el frontend), pueden ocurrir problemas de conexión, pérdida de mensajes o errores en la transmisión de datos.

Errores en MQTT:

La comunicación mediante MQTT puede fallar por diversas razones, como la desconexión de la red o problemas con el bróker (HiveMQ). Para manejar estos errores, es necesario configurar un sistema de reconexión automática y registrar los errores para poder analizarlos.

1. Reconexión automática: Cuando la conexión con el bróker se pierde, el cliente MQTT debe intentar reconectarse automáticamente.
2. Manejo de excepciones: Los errores en la recepción de mensajes deben ser capturados y gestionados adecuadamente [64].

```python
def on_disconnect(client, userdata, rc):
    print(f"Desconectado con código {rc}")
    if rc != 0:
        print("Reconectando...")
        client.reconnect()  # Intentar reconectar en caso de desconexión inesperada
```

Código 44. Ejemplo de código para reconexión automática en MQTT.

Errores en WebSocket (Django Channels):

La comunicación en tiempo real mediante WebSockets también puede enfrentar errores debido a desconexiones inesperadas o problemas con los datos recibidos. Es crucial manejar estos errores para evitar que la aplicación se detenga.

1. Reconexión automática: Si se pierde la conexión WebSocket, el cliente debe intentar reconectarse.
2. Manejo de excepciones: Cuando se reciben datos que no son válidos o cuando hay un error en el procesamiento de los mensajes, este debe ser capturado y manejado adecuadamente [32].

```python
from channels.generic.websocket import AsyncWebsocketConsumer
import json

class SensorDataConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.room_group_name = 'sensor_data'
        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()

    async def disconnect(self, close_code):
        await self.channel_layer.group_discard(
            self.room_group_name,
            self.channel_name
        )

    async def receive(self, text_data):
        try:
            data = json.loads(text_data)
            await self.send(text_data=json.dumps({
                'sensor_data': data
            }))
        except Exception as e:
            print(f"Error procesando los datos: {e}")
            await self.send(text_data=json.dumps({
                'error': 'Error al procesar los datos.'
            }))
```

Código 45. Ejemplo de código para manejar errores en Django Channels.

Validaciones de los Datos de los Sensores y las Configuraciones de los Dispositivos

Las validaciones son fundamentales para asegurar que los datos recibidos de los sensores sean correctos y que las configuraciones del sistema sean adecuadas. De

lo contrario, el sistema podría tomar decisiones incorrectas, como activar la bomba de agua en condiciones inadecuadas.

Validación de Datos de los Sensores:

Los datos de los sensores, como la temperatura o la humedad del suelo, deben estar dentro de rangos predefinidos. Se debe verificar que los datos sean consistentes y correctos [67].

```
from django.core.exceptions import ValidationError

def validate_sensor_data(data):
    if data['humidity'] < 0 or data['humidity'] > 100:
        raise ValidationError("La humedad debe estar entre 0 y 100.")
    if data['temperature'] < -10 or data['temperature'] > 50:
        raise ValidationError("La temperatura debe estar entre -10°C y 50°C.")
    return True
```

Código 46. Ejemplo de validación de datos de sensores.

Validación de Configuraciones de Dispositivos:

Las configuraciones de los sensores, como los umbrales de activación de la bomba de agua, deben ser validadas para asegurar que estén dentro de valores razonables.

```
def validate_sensor_configuration(config):
    if config['min_humidity'] < 0 or config['max_humidity'] > 100:
        raise ValidationError("Los umbrales de humedad deben estar entre 0 y
100.")
    if config['min_temperature'] < -10 or config['max_temperature'] > 50:
        raise ValidationError("Los umbrales de temperatura deben estar entre -10°C
y 50°C.")
    return True
```

Código 47. Ejemplo de validación de configuraciones de dispositivos.

Estas validaciones permiten que el sistema funcione de manera robusta, asegurando que solo se ejecuten acciones válidas y que los datos sean confiables [42].

B.2 Tareas automatizadas

Definición de Tareas Periódicas con Celery Beat

Una de las características de Celery Beat es que permite programar tareas que se ejecutan de manera periódica. En el contexto de este sistema, podemos tener tareas como la recolección periódica de datos de los sensores o la activación de dispositivos (como la bomba de agua).

Definición de una tarea periódica:

Supongamos que queremos que el sistema recoja los datos de los sensores cada 10 minutos. Para ello, definimos una tarea en Celery que será ejecutada periódicamente.

```python
# tasks.py
from celery import shared_task
from .models import SensorData
from .utils import get_sensor_data

@shared_task
def fetch_sensor_data():
    """
    Tarea periódica que obtiene los datos de los sensores y los guarda en la base
de datos.
    """
    data = get_sensor_data()  # Función que obtiene los datos de los sensores
    SensorData.objects.create(value=data['value'], timestamp=data['timestamp'])
    print('Datos de sensores recogidos y almacenados')
```

Código 48. Ejemplo de tarea Celery para la recolección de datos.

Esta tarea **`fetch_sensor_data`** se encarga de obtener los datos de los sensores (a través de una función externa **`get_sensor_data`**) y almacenarlos en la base de datos.

Programación de la tarea con Celery Beat:

Para que esta tarea se ejecute cada 10 minutos, la configuramos en Celery Beat. A continuación, se muestra cómo hacerlo:

```python
# settings.py
from celery.schedules import crontab

CELERY_BEAT_SCHEDULE = {
    'fetch-sensor-data-every-10-minutes': {
        'task': 'myapp.tasks.fetch_sensor_data',  # Nombre de la tarea
        'schedule': crontab(minute='*/10'),  # Ejecutar cada 10 minutos
    },
}
```

Código 49. Programación de una tarea con Celery Beat.

La línea **`crontab(minute='*/10')`** indica que la tarea se debe ejecutar cada 10 minutos [50].

Ejecución de Tareas Periódicas

Cuando se ejecuta el servidor de Celery con Celery Beat, el programador manejará las tareas periódicas automáticamente. Aquí hay una forma básica de ejecutar Celery y Celery Beat:

Ejecutando Celery con Celery Beat:

```
celery -A myproject worker --loglevel=info
```

```
celery -A myproject beat --loglevel=info
```

El comando worker se encarga de ejecutar las tareas, y beat gestiona la programación de las tareas periódicas [50].

Monitoreo y Gestión de Tareas

Es importante monitorear y gestionar las tareas que se ejecutan en segundo plano. Celery ofrece herramientas de monitoreo que permiten ver qué tareas se están ejecutando, si alguna ha fallado o si hay tareas pendientes. Además, se pueden configurar tareas de forma que se puedan ejecutar de manera más eficiente en entornos de producción.

Ejemplo de monitoreo de tareas:

Celery permite ver el estado de las tareas ejecutadas utilizando herramientas de monitoreo, como Flower, una herramienta de monitoreo en tiempo real para Celery.

```
pip install flower
```

```
celery -A myproject flower
```

Esto abrirá una interfaz web para monitorear las tareas de Celery en tiempo real [34].

Manejo de Errores en Tareas Periódicas

Como las tareas de Celery se ejecutan en segundo plano, es crucial gestionar los errores que puedan ocurrir durante la ejecución. Algunos mecanismos de manejo de errores incluyen:

- Reintentos automáticos: Si una tarea falla, Celery puede configurarse para reintentarla automáticamente después de un cierto periodo.
- Notificación de errores: Se pueden configurar alertas para notificar a los administradores si una tarea crítica falla.

```python
from celery import Celery
from celery.exceptions import MaxRetriesExceededError

@shared_task(bind=True, max_retries=3)
def fetch_sensor_data(self):
    try:
        data = get_sensor_data()  # Obtener datos
        SensorData.objects.create(value=data['value'],
timestamp=data['timestamp'])
    except Exception as e:
        raise self.retry(exc=e, countdown=60)  # Reintentar después de 60 segundos
```

Código 50. Ejemplo de reintentos automáticos en Celery.

En este caso, Celery intentará ejecutar la tarea hasta 3 veces si ocurre un error [34].

B.3 API y comunicación en tiempo real

Implementación de WebSocket en Django Channels

Django Channels proporciona una manera sencilla de gestionar WebSockets mediante el uso de consumers. Un consumer es responsable de recibir las conexiones WebSocket y enviar/recibir mensajes entre el cliente y el servidor.

Configuración del WebSocket Consumer en Django

A continuación, se muestra cómo crear un consumer en Django Channels para manejar las conexiones WebSocket, recibir mensajes y enviar actualizaciones a los clientes.

1. Instalación de Django Channels:

Primero, es necesario instalar Django Channels si aún no se ha hecho:

```
pip install channels
```

2. Configuración del consumidor WebSocket:

```python
# consumers.py
from channels.generic.websocket import AsyncWebsocketConsumer
import json

class SensorDataConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        # Establecemos el nombre del grupo de canales
        self.room_group_name = 'sensor_data'

        # Unirse al grupo de canales
        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )

        # Aceptamos la conexión WebSocket
        await self.accept()

    async def disconnect(self, close_code):
        # Salir del grupo de canales
        await self.channel_layer.group_discard(
            self.room_group_name,
            self.channel_name
        )

    async def receive(self, text_data):
        # Procesamos los mensajes entrantes
        data = json.loads(text_data)
        # Enviar los datos al grupo de canales
        await self.send(text_data=json.dumps({
            'sensor_data': data
        }))

    # Método para enviar datos al WebSocket del cliente
    async def send_sensor_data(self, event):
        # Enviamos el mensaje al cliente WebSocket
        await self.send(text_data=json.dumps({
            'sensor_data': event['data']
        }))
```

Código 51. Configuración del consumidor WebSocket.

Explicación:

- **`connect`**: Establece la conexión WebSocket y la añade a un grupo de canales (en este caso, "**`sensor_data`**").
- **`disconnect`**: Cuando el cliente se desconecta, se elimina del grupo de canales.

- **`receive`**: Recibe los mensajes del cliente WebSocket (en este caso, los datos de los sensores), y los transmite de vuelta al cliente a través de **`send_sensor_data`**.
- **`send_sensor_data`**: Este método es utilizado para enviar datos al cliente. El backend puede llamar a este método para enviar actualizaciones al frontend en tiempo real [32].

Configuración de Channels Layer (Canal de Comunicación)

Django Channels utiliza un channel layer para gestionar la comunicación entre los consumidores y distribuir los mensajes a través de grupos. Para configurarlo, utilizamos Redis como un backend para el channel layer.

Instalación de Redis:

Para instalar Redis, ejecutamos el siguiente comando:

```
pip install channels_redis
```

Configuración en `settings.py`:

```
# settings.py
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            "hosts": [('127.0.0.1', 6379)],
        },
    },
}
```

Código 52. Configuración en `settings.py`.

Con esta configuración, Django Channels utilizará Redis como backend para gestionar las conexiones y las tareas asíncronas [32].

Envío de Datos de los Sensores en Tiempo Real

Una vez que el backend recibe los datos de los sensores (por ejemplo, a través de MQTT como se mostró en la sección anterior), puede enviar estos datos en tiempo real al frontend a través de WebSockets.

Ejemplo de cómo enviar los datos de sensores al frontend:

```python
# consumers.py (continuación)
from .models import SensorData

async def send_sensor_data_to_frontend(self):
    # Obtener datos más recientes de la base de datos
    sensor_data = SensorData.objects.latest('timestamp')
    data = {
        'humidity': sensor_data.humidity,
        'temperature': sensor_data.temperature,
    }

    # Enviar los datos al cliente WebSocket
    await self.send(text_data=json.dumps({
        'sensor_data': data
    }))
```

Código 53. Ejemplo de cómo enviar los datos de sensores al frontend.

En este ejemplo, el backend consulta la base de datos para obtener los datos más recientes de los sensores y los envía al frontend a través de la conexión WebSocket en tiempo real [32].

Creación de Tablas en TimescaleDB

A continuación, se describe cómo crear las tablas en TimescaleDB, basadas en el modelo de datos proporcionado:

```sql
-- Crear la tabla de tipos de sensores
CREATE TABLE sensors_sensor_type (
    id UUID PRIMARY KEY,
    name VARCHAR(100),
    unit VARCHAR(50)
);

-- Crear la tabla de sensores
CREATE TABLE sensors_sensor (
    id UUID PRIMARY KEY,
    name VARCHAR(100),
    min_value FLOAT,
    max_value FLOAT,
    location VARCHAR(255),
    description TEXT,
    type_id UUID,
    FOREIGN KEY (type_id) REFERENCES sensors_sensor_type(id)
);

-- Crear la tabla de datos de sensores
CREATE TABLE sensors_sensordata (
    time TIMESTAMPTZ(6) PRIMARY KEY,
    value FLOAT,
    sensor_id UUID,
    FOREIGN KEY (sensor_id) REFERENCES sensors_sensor(id)
);

-- Crear la tabla de notificaciones
CREATE TABLE sensors_notification (
    id UUID PRIMARY KEY,
    status VARCHAR(50),
    message TEXT,
    timestamp TIMESTAMPTZ(6),
    sensor_id UUID,
    FOREIGN KEY (sensor_id) REFERENCES sensors_sensor(id)
);
```

Código 54. Creación de Tablas en TimescaleDB.

Explicación:

- **sensors_sensor_type** almacena los tipos de sensores, como temperatura, humedad, etc., con la unidad correspondiente.
- **sensors_sensor** almacena los sensores individuales, con información como su nombre, ubicación, valores mínimos y máximos, y su tipo (relacionado con la tabla **sensors_sensor_type**).
- **sensors_sensordata** almacena las mediciones de los sensores, incluyendo la marca de tiempo y el valor medido.

- **`sensors_notification`** almacena las notificaciones generadas por los sensores (por ejemplo, cuando los valores de los sensores superan ciertos umbrales) [40].

Creación de Hypertable para Datos de Sensores

Para aprovechar las capacidades de TimescaleDB y optimizar la consulta de series temporales, convertimos la tabla **`sensors_sensordata`** en una hypertable. Las hypertables permiten que los datos se particionen automáticamente en función del tiempo, lo que mejora el rendimiento de las consultas sobre grandes volúmenes de datos.

```
-- Crear una hypertable para almacenar los datos de los sensores
SELECT create_hypertable('sensors_sensordata', 'time');
```

Código 55. Código SQL para convertir `sensors_sensordata` en una hypertable.

Esto convierte la tabla **`sensors_sensordata`** en una hypertable, lo que permite manejar de manera eficiente grandes cantidades de datos que se generan con frecuencia [40].

Inserción y Consulta de Datos

Ahora que las tablas están configuradas, podemos insertar y consultar los datos de manera eficiente. A continuación, se muestran ejemplos de cómo insertar datos en la base de datos y cómo realizar consultas sobre los datos almacenados.

Inserción de Datos:

```python
from .models import SensorData

def insert_sensor_data(sensor_id, value):
    timestamp = timezone.now()  # Obtener la marca de tiempo actual
    sensor_data = SensorData(
        sensor_id=sensor_id,
        time=timestamp,
        value=value
    )
    sensor_data.save()
```

Código 56. Ejemplo de inserción de datos.

Consultas de Datos:

1. Obtener los últimos 10 registros de datos de sensores:

```sql
SELECT * FROM sensors_sensordata
ORDER BY time DESC
LIMIT 10;
```

Código 57. Consulta SQL para obtener los últimos registros de datos sensados.

2. Obtener el promedio de las mediciones de un sensor en los últimos 30 minutos:

```sql
SELECT avg(value)
FROM sensors_sensordata
WHERE sensor_id = 'sensor-id-aqui'
AND time > NOW() - INTERVAL '30 minutes';
```

Código 58. Cálculo del promedio de lecturas de un sensor en los últimos 30 minutos.

Escalabilidad y Gestión del Almacenamiento

TimescaleDB permite gestionar grandes volúmenes de datos de manera eficiente, y proporciona herramientas para implementar políticas de retención de datos. Esto es útil para eliminar datos antiguos que ya no son relevantes, optimizando el almacenamiento.

Política de Retención de Datos:

```
-- Eliminar los datos de sensores más antiguos que 6 meses
SELECT drop_chunks('sensors_sensordata', INTERVAL '6 months');
```

Código 59. Configuración de la política de retención de datos.

Esto eliminará de forma automática los datos más antiguos de la tabla **`sensors_sensordata`**, lo que ayuda a mantener el rendimiento de la base de datos a medida que el volumen de datos crece [40].